

ПРЕДИСЛОВИЕ

Около трех с половиной десятилетий минуло с тех пор, как в педвузах введено в качестве учебной дисциплины программирование для ЭВМ.

За все это время педвузовские кафедры программирования и вычислительной математики, как и образованные позднее на их основе кафедры информатики и вычислительной техники, не были избалованы своевременным появлением специальных учебных изданий. При колоссальной скорости изменений в самом предмете, всегда существенно превышавшей скорость центральных издательских механизмов, специально ориентированные на программы педвузов книги выходили не чаще, чем раз в десятилетие - едва ли не соразмерно скорости смены поколений ЭВМ. Если к этому добавить непреходящую особенность книг о компьютерах и программах к ним устаревать еще в типографии, то можно с грустью утверждать, что нормальных учебников по Computer Science для педвузов пока - увы! - так и не было. И можно ли считать утешением тот безрадостный факт, что уровень компьютерного обеспечения большинства педвузов обычно с трудом «дотягивал» до содержания полустарых учебных пособий?

В наши дни многое изменилось. Во-первых, на месте узкого учебного курса программирования в учебных планах педвузов давно уже обосновался куда более обширный и многоаспектный учебный предмет «Информатика». Да и положение с источниками, конечно, стало другим - полки книжных торговцев сегодня ломятся от изданий по информатике.

Однако преподавателю (а более всего студенту) специальная учебная книга, содержание и направленность которой отвечают заданному учебному плану и программе (или, как сейчас говорят, стандарту образования), все-таки очень нужна. Сделать такую книгу непросто: с одной стороны - пугающая широта и вечная необустроенность предмета информатики, а с другой - динамичность, неуспокоенность самих стандартов педобразования, равно как и школьных стандартов, которые тоже именно сейчас находятся в состоянии активного переосмысления. Эти взаимосвязанные предпосылки достаточно долгое время оставляли мало надежд на то, что найдутся отважные авторы (преисполненные профессионального долга, а вовсе не амбициозного желания прославиться), которые возьмутся за такую работу и сделают ее. Те, кто держит в руках эту книгу, могут убедиться, что за дело взялись наиболее авторитетные и хорошо известные в кругу педагогов-информатиков ученые. Мне же приятно осознавать, что с этими замечательными людьми и специалистами нас не один год связывало Учебно-методическое объединение педагогических учебных заведений России по информатизации образования, в котором и родилась инициатива написания этого учебного пособия.

М. П. Лапчик, Действительный член
Академии информатизации
образования, профессор

ВВЕДЕНИЕ

Уважаемый читатель!

Эта книга - учебник информатики - прежде всего для тех, кто избрал областью профессиональной деятельности ее Преподавание. Такой выбор налагает двойную ответственность: знать все аспекты данного предмета на достаточно высоком уровне и уметь рассказать о нем, научить других.

Авторы надеются, что учебник будет полезен не только студентам педагогических вузов и «классических» университетов, избравших обучение информатике сферой своей будущей профессиональной деятельности, но и практикующим учителям информатики для самообразования и повышения квалификации.

Эта книга может быть полезна и многим другим, изучающим информатику. В первом приближении всех тех, кого информатика интересует, можно разбить на три категории. Представители первой, самой многочисленной, довольствуются несколькими широко распространенными информационными технологиями: обработки текстов и графической информации, работы в сетях и т.д., причем им не надо знать, как все это происходит. Таких можно назвать «принципиальными пользователями» (без малейших намерений обидеть, у большинства из них просто иная сфера профессиональных интересов). Для них наш учебник содержит слишком много теоретических сведений и недостаточно указаний о «кнопочной» стороне дела, за которой мы часто отсылаем читателей к специальной литературе. Вторая категория состоит из тех, кто интересуется не только реализацией информационных технологий, но и ответами на вопросы, «как это делается и почему именно так». Хочется верить, что на большую часть таких вопросов данный учебник дает ответ. Наконец, третья категория - будущие профессионалы, которые намереваются работать в одной из сфер информатики, - разработчики программного обеспечения (трансляторов, издательских систем, экспертных систем и т.п.), сетевики, администраторы крупных информационных систем и т.д. Для них по направлению избранной специализации, скорее всего, нужны более глубокие знания, а данное пособие может стать книгой для первого чтения и общим обзором, ибо трудно профессионально углубиться во все сферы информатики в равной мере.

Современная информатика очень велика по объему и очень динамична. Если изучаемые в вузах курсы математики, лингвистики, химии и большинства других наук практически не изменятся на протяжении многих лет будущей профессиональной деятельности сегодняшнего студента, то в информатике это полностью оформившееся ядро сравнительно невелико. Как же тогда быть с ее изучением?

Прежде всего, следует определиться, что такое информатика. В понимании некоторых людей это есть совокупность приемов и методов работы с компьютерами. На самом деле это не так: компьютеры являются лишь техническим средством, с помощью которого информатика реализует свой прикладной пользовательский аспект - правда, средством столь сложным и интересным, что оно способно поглотить массу внимания не только специалистов в области компьютерных технологий, но и непрофессионалов.

Информатика - комплекс научно-практических дисциплин, изучающих все аспекты получения, хранения, преобразования, передачи и использования информации. Более детальное обсуждение содержания информатики, отражение разных точек зрения, проводится ниже, в главе 1. Однако, из какого бы определения ни исходить, все согласны с тем, что у современной информатики есть два взаимодополняющих аспекта - научный и технологический. Первый является более устойчивым, второй - весьма мобильным, хотя и в технологической части информатики есть вполне сформировавшееся ядро, которое мало подвержено изменениям.

Приведем примеры, опираясь на имеющийся у читателей опыт. Так, существует большое количество алгоритмических языков программирования, и допустим, что человеку, умеющему работать с Бейсиком или Паскалем, приходится браться за Си. Новая система обозначений, дополнительные возможности - на некоторое время это может полностью поглотить внимание, но постепенно приходит понимание: главное - навыки к алгоритмизации и структурированию данных, и если они есть, то кодирование алгоритмов на другом языке - дело не самое сложное. Или: истощены возможности привычного текстового редактора (или он просто вышел из моды - тоже бывает), и нужно переходить на новый. Если человек понимает принципы работы программ такого рода, имеет устойчивые навыки работы с одной из них, то освоить другую, даже с большими воз-

возможностями, обычно несложно. Подтверждается известная истина: образование - это то, что остается, когда детали знаний забываются.

Итак, главное при изучении информатики - освоить фундаментальные понятия каждой из ее областей, ориентироваться в их взаимосвязи, приобрести навыки практической работы с важнейшими техническими и программными средствами. Добавим, что от того, кто хочет стать учителем, требуется в определенном смысле больше, чем, скажем, от инженера-разработчика ЭВМ или программиста - преподаватель должен знать практически все разделы информатики, владеть ее техническими средствами на уровне высококвалифицированного пользователя и уметь передавать свои знания и навыки другим людям.

Есть еще один, чрезвычайно важный аспект подготовки будущего бакалавра или специалиста - научиться самостоятельно осваивать новые знания и навыки. Никакой учебник, и наш в том числе, не может содержать всех необходимых сведений - тем более что в вашу подготовку будут входить, кроме базовых дисциплин, спецкурсы, выполнение курсовых и выпускных работ. Для того чтобы вам в этом помочь, в учебнике приводятся справочные материалы, ссылки на книги, в которых обсуждаемые вопросы излагаются более детально. Если вы привыкнете пользоваться этим справочным аппаратом, дополнительной литературой, то никакие перемены в содержании курсов, которые вы сами будете в будущем преподавать, не страшны.

Данный учебник охватывает основные разделы современной информатики. Чрезвычайно важна теоретическая база, закладываемая в главе 1, - это прочно устоявшееся ядро нашей науки, которому не грозят существенные перемены.

Глава 2 посвящена необъятному вопросу о программном обеспечении ЭВМ. Ее цель - ознакомление с основными классами базового и прикладного программного обеспечения, фундаментальными принципами, заложенными в их основу. Разумеется, в ходе изучения этих вопросов предполагается выработка практических навыков пользования типичными программами каждого класса.

Глава 3 специально названа не «языки программирования», а «языки и методы программирования». Будущий преподаватель не может себе позволить ограничиться глубоким изучением одного-двух языков, какими бы мощными и популярными они ни были; ему необходим широкий кругозор в этих вопросах. Различные методологии («парадигмы») программирования, описанные в этой главе, отражают различные стили, способы человеческого мышления.

Без отчетливого понимания основ функционирования вычислительной техники (глава 4) нельзя всерьез освоить многие другие разделы информатики.

Нижним уровнем этого понимания, который от вас требуется, является уровень архитектуры ЭВМ, но и он весьма глубок и предполагает проникновение в новую понятийную область. Понимание это невозможно без ознакомления с программированием на уровне команд микропроцессора. Необходимо также понимать принципы функционирования и уметь работать со многочисленными внешними устройствами ЭВМ.

Компьютерные сети и телекоммуникации - весьма популярный сегодня раздел прикладной информатики. Осознав возможность передачи текстовой, графической и иной информации на любые расстояния в кратчайшее время, получив экономическую возможность вовлечь в этот процесс миллионы людей, человечество буквально рванулось в мир компьютерных сетей. Этим вопросам посвящена глава 5.

В главе 6 рассказывается об информационных системах, без которых невозможно представить себе современную информатику. Базы и банки данных, информационно-поисковые системы и автоматизированные управляющие системы, равно как и многие другие виды информационных систем, обсуждаются в этой главе. Много внимания уделяется, в частности, информационным системам в образовании.

Могучей информационной технологией, которой полностью посвящена глава 7, является компьютерное моделирование - интегративная дисциплина, включающая выходы в самые различные науки. Моделирование является важным методом человеческого познания, в котором компьютеры выступают как мощное техническое средство.

Курс информатики, который охватывает данное учебное пособие, должен включать в себя большое число практических занятий и лабораторных работ. В ходе их выполнения вы приобретете навыки в пользовании ЭВМ, в работе со многими программными продуктами, в создании относительно несложных программ, освоитесь в современных информационных технологиях. Важную

роль при этом играют смежные дисциплины предметной подготовки - в первую очередь, математика, физика. В некоторых их разделах будут существенно углублены сведения, непосредственно примыкающие к информатике, - по математической логике и теории алгоритмов, по электронике и физическим основам функционирования ЭВМ и другим.

Последовательность изучения регулярного курса информатики может отличаться от той, в которой расположен материал в данной книге. Информатика не столь формализована как, например, математика, и многие разделы информатики при изучении можно менять местами. Поэтому, встретив в книге новое для вас понятие, которое не объяснено в данном месте, поищите разъяснение в другом разделе или в справочнике, толковом словаре, которые хорошо бы всегда иметь на столе.

При отборе материала для пособия основой послужили важнейшие нормативные документы - Государственные образовательные стандарты высшего профессионального образования, определяющие требования к обязательному минимуму содержания и уровню подготовки бакалавров образования по направлению «Естественные науки», профилю «Информатика», и специалистов - учителей информатики (тем более, что авторы данного пособия участвовали в разработке предметного блока стандарта подготовки учителя информатики). Хотя указанные стандарты должны периодически пересматриваться, их основа наверняка сохранится.

Процитируем общие требования, предъявляемые Стандартами к знаниям и умениям бакалавра образования по профилю «Информатика» и специалиста -учителя информатики по дисциплинам предметной подготовки.

Бакалавр должен

- иметь целостное представление об информатике как науке, ее месте в современном мире и в системе наук;
- владеть системой знаний о теоретических основах информатики;
- знать программное обеспечение ЭВМ;
- знать устройство ЭВМ, тенденцию развития архитектуры ЭВМ;
- обладать навыками программирования на нескольких проблемно- и машинно-ориентированных языках, отладки и тестирования программ;
- иметь навыки работы с ЭВМ нескольких типов, различными вспомогательными устройствами, с системными и прикладными программными средствами общего назначения;
- владеть методами работы в локальных сетях и системах телекоммуникаций;
- владеть методологией построения математических моделей и их компьютерных реализации, знать конкретные математические модели в различных областях.

Авторы учебного пособия - заведующие кафедрами информатики Воронежского, Красноярского и Пермского педагогических университетов, много лет отдавшие подготовке учителей информатики. В этой книге их коллективный опыт. В работе над пособием авторам оказывали помощь их коллеги. Особо следует отметить доцента Е.А.Еремина и А.П.Шестакова, которые приняли большое участие в написании глав 4 и 7, соответственно. Желаем успеха читателям!

ЧАСТЬ ПЕРВАЯ

ГЛАВА 1

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

Нет столь великой вещи, которую не превзошла бы еще большая.

Козьма Прутков

ВВЕДЕНИЕ

Практически в каждой науке есть фундамент, без которого ее прикладные аспекты лишены основ. Для математики такой фундамент составляют теория множеств, теория чисел, математическая логика и некоторые другие разделы; для физики - это основные законы классической и квантовой механики, статистической физики, релятивистской теории; для химии - периодический закон, его теоретические основы и т.д. Можно, конечно, научиться считать и пользоваться калькулятором, даже не подозревая о существовании указанных выше разделов математики, делать химические анализы без понимания существа химических законов, но при этом не следует думать, что знаешь математику или химию. Примерно то же с информатикой: можно изучить несколько программ и даже освоить некоторое ремесло, но это отнюдь не вся информатика, точнее, даже не самая главная и интересная ее часть.

Теоретические основы информатики - пока не вполне сложившийся, устоявшийся раздел науки. Он возникает на наших глазах, что делает его особенно интересным - нечасто мы наблюдаем и даже можем участвовать в рождении новой науки! Как и теоретические разделы других наук, теоретическая информатика формируется в значительной мере под влиянием потребностей обучения информатике.

Теоретическая информатика - наука математизированная. Она складывается из ряда разделов математики, которые прежде казались мало связанными друг с другом: теории автоматов и теории алгоритмов, математической логики, теории формальных языков и грамматик, реляционной алгебры, теории информации и др. Теоретическая информатика старается методами точного анализа ответить на основные вопросы, возникающие при работе с информацией, например вопрос о количестве информации, сосредоточенной в той или иной информационной системе, наиболее рациональной организации таких систем для хранения и поиска информации, а также о существовании и свойствах алгоритмов преобразования информации. Конструкторы устройств хранения данных проявляют чудеса изобретательности, увеличивая объем и плотность хранения данных на дисках, но в основе этой деятельности лежат теория информации и теория кодирования. Для решения прикладных задач существуют замечательные программы, но для того, чтобы грамотно поставить прикладную задачу, привести ее к виду, который подвластен компьютеру, надо знать основы информационного и математического моделирования и т.д. Только освоив эти разделы информатики, можно считать себя специалистом в этой науке. Другое дело - с какой глубиной осваивать; многие разделы теоретической информатики достаточно сложны и требуют основательной математической подготовки. В большинстве случаев они излагаются ниже, скорее, в ознакомительном порядке, с целью составить о них отчетливое представление.

§ 1. ИНФОРМАТИКА КАК НАУКА И КАК ВИД ПРАКТИЧЕСКОЙ ДЕЯТЕЛЬНОСТИ

1.1. ИСТОРИЯ РАЗВИТИЯ ИНФОРМАТИКИ

Информатика - молодая научная дисциплина, изучающая вопросы, связанные с поиском, сбором, хранением, преобразованием и использованием информации в самых различных сферах человеческой деятельности. Генетически информатика связана с вычислительной техникой, компьютерными системами и сетями, так как именно компьютеры позволяют порождать, хранить и автоматически перерабатывать информацию в таких количествах, что научный подход к информационным процессам становится одновременно необходимым и возможным.

До настоящего времени толкование термина «информатика» (в том смысле как он используется в современной научной и методической литературе) еще не является установившимся и общепринятым. Обратимся к истории вопроса, восходящей ко времени появления электронных вычислительных машин.

После второй мировой войны возникла и начала бурно развиваться кибернетика как наука об общих закономерностях в управлении и связи в различных системах: искусственных, биологических, социальных. Рождение кибернетики принято связывать с опубликованием в 1948 г. американским математиком Норбертом Винером, ставшей знаменитой, книги «Кибернетика или управление и связь в животном и машине». В этой работе были показаны пути создания общей теории управления и заложены основы методов рассмотрения проблем управления и связи для различных систем с единой точки зрения. Развиваясь одновременно с развитием электронно-вычислительных машин, кибернетика со временем превращалась в более общую науку о преобразовании информации. Под информацией в кибернетике понимается любая совокупность сигналов, воздействий или сведений, которые некоторой системой воспринимаются от окружающей среды (входная информация X), выдаются в окружающую среду (выходная информация Y), а также хранятся в себе (внутренняя, внутрисистемная информация Z), рис. 1.1.

Развитие кибернетики в нашей стране встретило идеологические препятствия. Как писал академик А.И.Берг, «... в 1955-57 гг. и даже позже в нашей литературе были допущены грубые ошибки в оценке значения и возможностей кибернетики. Это нанесло серьезный ущерб развитию науки в нашей стране, привело к задержке в разработке многих теоретических положений и даже самих электронных машин». Достаточно сказать, что еще в философском словаре 1959 года издания кибернетика характеризовалась как «буржуазная лженаука». Причиной этому послужили, с одной стороны, недооценка новой бурно развивающейся науки отдельными учеными «классического» направления, с другой - неумеренное пустословие тех, кто вместо активной разработки конкретных проблем кибернетики в различных областях спекулировал на полуфантастических прогнозах о безграничных возможностях кибернетики, дискредитируя тем самым эту науку.

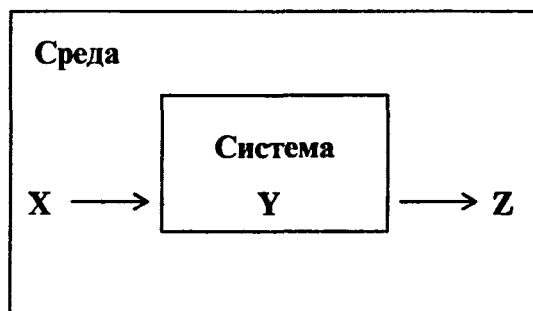


Рис. 1.1. Общая схема обмена информацией между системой и внешней средой

Дело к тому же осложнялось тем, что развитие отечественной кибернетики на протяжении многих лет сопровождалось серьезными трудностями в реализации крупных государственных проектов, например, создания автоматизированных систем управления (АСУ). Однако за это время удалось накопить значительный опыт создания информационных систем и систем управления технико-экономическими объектами. Требовалось выделить из кибернетики здоровее научное и техническое ядро и консолидировать силы для развития нового движения к давно уже стоящим глобальным целям.

Подойдем сейчас к этому вопросу с терминологической точки зрения. Вскоре вслед за появлением термина «кибернетика» в мировой науке стало использоваться англоязычное «Computer Science», а чуть позже, на рубеже шестидесятых и семидесятых годов, французы ввели получивший сейчас широкое распространение термин «Informatique». В русском языке раннее употребление термина «информатика» связано с узко-конкретной областью изучения структуры и общих свойств научной информации, передаваемой посредством научной литературы. Эта информационно-аналитическая деятельность, совершенно необходимая и сегодня в библиотечном деле, книгоиздании и т.д., уже давно не отражает современного понимания информатики. Как отмечал академик А.П.Ершов, в современных условиях термин информатика «вводится в русский язык в но-

вом и куда более широком значении - как название фундаментальной естественной науки, изучающей процессы передачи и обработки информации. При таком толковании информатика оказывается более непосредственно связанной с философскими и общенаучными категориями, проясняется и ее место в кругу "традиционных" академических научных дисциплин».

Попытку определить, что же такое современная информатика, сделал в 1978 г. Международный конгресс по информатике: «Понятие информатики охватывает области, связанные с разработкой, созданием, использованием и материально-техническим обслуживанием систем обработки информации, включая машины, оборудование, математическое обеспечение, организационные аспекты, а также комплекс промышленного, коммерческого, административного и социального воздействия».

1.2. ИНФОРМАТИКА КАК ЕДИНСТВО НАУКИ И ТЕХНОЛОГИИ

Информатика - отнюдь не только «чистая наука». У нее, безусловно, имеется научное ядро, но важная особенность информатики - широчайшие приложения, охватывающие почти все виды человеческой деятельности: производство, управление, науку, образование, проектные разработки, торговлю, финансовую сферу, медицину, криминалистику, охрану окружающей среды и др. И, может быть, главное из них - совершенствование социального управления на основе новых информационных технологий.

Как наука, информатика изучает общие закономерности, свойственные информационным процессам (в самом широком смысле этого понятия). Когда разрабатываются новые носители информации, каналы связи, приемы кодирования, визуального отображения информации и многое другое, конкретная природа этой информации почти не имеет значения. Для разработчика системы управления базами данных (СУБД) важны общие принципы организации и эффективность поиска данных, а не то, какие конкретно данные будут затем заложены в базу многочисленными пользователями. Эти общие закономерности есть предмет информатики как науки.

Объектом приложений информатики являются самые различные науки и области практической деятельности, для которых она стала непрерывным источником самых современных технологий, называемых часто «новые информационные технологии» (НИТ). Многообразные информационные технологии, функционирующие в разных видах человеческой деятельности (управлении производственным процессом, проектировании, финансовых операциях, образовании и т.п.), имея общие черты, в то же время существенно различаются между собой.

Перечислим наиболее впечатляющие реализации информационных технологий, используя, ставшие традиционными, сокращения.

АСУ - автоматизированные системы управления - комплекс технических и программных средств, которые во взаимодействии с человеком организуют управление объектами в производстве или общественной сфере. Например, в образовании используются системы АСУ-ВУЗ.

АСУТП - автоматизированные системы управления технологическими процессами. Например, такая система управляет работой станка с числовым программным управлением (ЧПУ), процессом запуска космического аппарата и т.д.

АСНИ - автоматизированная система научных исследований - программно-аппаратный комплекс, в котором научные приборы сопряжены с компьютером, вводят в него данные измерений автоматически, а компьютер производит обработку этих данных и представление их в наиболее удобной для исследователя форме.

АОС - автоматизированная обучающая система. Есть системы, помогающие учащимся осваивать новый материал, производящие контроль знаний, помогающие преподавателям готовить учебные материалы и т.д.

САПР-система автоматизированного проектирования - программно-аппаратный комплекс, который во взаимодействии с человеком (конструктором, инженером-проектировщиком, архитектором и т.д.) позволяет максимально эффективно проектировать механизмы, здания, узлы сложных агрегатов и др.

Упомянем также диагностические системы в медицине, системы организации продажи билетов, системы ведения бухгалтерско-финансовой деятельности, системы обеспечения редакционно-издательской деятельности - спектр применения информационных технологий чрезвычайно широк.

С развитием информатики возникает вопрос о ее взаимосвязи и разграничении с кибернетикой. При этом требуется уточнение предмета кибернетики, более строгое его толкование. Информатика и кибернетика имеют много общего, основанного на концепции управления, но имеют и объективные различия. Один из подходов разграничения информатики и кибернетики - отнесение к области информатики исследований информационных технологий не в любых кибернетических системах (биологических, технических и т.д.), а только в социальных системах. В то время как за кибернетикой сохраняются исследования общих законов движения информации в производственных системах, информатика, опираясь на этот теоретический фундамент, изучает конкретные способы и приемы переработки, передачи, использования информации. Впрочем, многим современным ученым такое разделение представляется искусственным, и они просто считают кибернетику одной из составных частей информатики.

1.3. СТРУКТУРА СОВРЕМЕННОЙ ИНФОРМАТИКИ

Оставляя в стороне прикладные информационные технологии, опишем составные части «ядра» современной информатики. Каждая из этих частей может рассматриваться как относительно самостоятельная научная дисциплина; взаимоотношения между ними примерно такие же, как между алгеброй, геометрией и математическим анализом в классической математике - все они хоть и самостоятельные дисциплины, но, несомненно, части одной науки.

Теоретическая информатика - часть информатики, включающая ряд математических разделов. Она опирается на математическую логику и включает такие разделы как теория алгоритмов и автоматов, теория информации и теория кодирования, теория формальных языков и грамматик, исследование операций и другие. Этот раздел информатики использует математические методы для общего изучения процессов обработки информации.

Вычислительная техника - раздел, в котором разрабатываются общие принципы построения вычислительных систем. Речь идет не о технических деталях и электронных схемах (это лежит за пределами информатики как таковой), а о принципиальных решениях на уровне, так называемой, **архитектуры** вычислительных (компьютерных) систем, определяющей состав, назначение, функциональные возможности и принципы взаимодействия устройств. Примеры принципиальных, ставших классическими решений в этой области - неймановская архитектура компьютеров первых поколений, шинная архитектура ЭВМ старших поколений, архитектура параллельной (многопроцессорной) обработки информации.

Программирование - деятельность, связанная с разработкой систем программного обеспечения. Здесь отметим лишь основные разделы современного программирования: создание системного программного обеспечения и создание прикладного программного обеспечения. Среди системного - разработка новых языков программирования и компиляторов к ним, разработка интерфейсных систем (пример - общеизвестная операционная оболочка и система Windows). Среди прикладного программного обеспечения общего назначения самые популярные - система обработки текстов, электронные таблицы (табличные процессоры), системы управления базами данных. В каждой области предметных приложений информатики существует множество специализированных прикладных программ более узкого назначения.

Информационные системы - раздел информатики, связанный с решением вопросов по анализу потоков информации в различных сложных системах, их оптимизации, структурировании, принципах хранения и поиска информации. Информационно-справочные системы, информационно-поисковые системы, гигантские современные глобальные системы хранения и поиска информации (включая широко известный Internet) в последнее десятилетие XX века привлекают внимание все большего круга пользователей. Без теоретического обоснования принципиальных решений в океане информации можно просто захлебнуться. Известным примером решения проблемы на глобальном уровне может служить гипертекстовая поисковая система WWW, а на значительно более низком уровне - справочная система, к услугам которой мы прибегаем, набрав телефонный номер 09'.

Искусственный интеллект - область информатики, в которой решаются сложнейшие проблемы, находящиеся на пересечении с психологией, физиологией, лингвистикой и другими науками. Как научить компьютер мыслить подобно человеку? - Поскольку мы далеко не все знаем о том, как мыслит человек, исследования по искусственному интеллекту, несмотря на полувековую

историю, все еще не привели к решению ряда принципиальных проблем. Основные направления разработок, относящихся к этой области - моделирование рассуждений, компьютерная лингвистика, машинный перевод, создание экспертных систем, распознавание образов и другие. От успехов работ в области искусственного интеллекта зависит, в частности, решение такой важнейшей прикладной проблемы как создание интеллектуальных интерфейсных систем взаимодействия человека с компьютером, благодаря которым это взаимодействие будет походить на межчеловеческое и станет более эффективным.

1.4. МЕСТО ИНФОРМАТИКИ В СИСТЕМЕ НАУК

Рассмотрим место науки информатики в традиционно сложившейся системе наук (технических, естественных, гуманитарных и т.д.). В частности, это позволило бы найти место общеобразовательного курса информатики в ряду других учебных предметов.

Напомним, что по определению А.П.Ершова информатика- «фундаментальная естественная наука». Академик Б.Н.Наумов определял информатику «как естественную науку, изучающую общие свойства информации, процессы, методы и средства ее обработки (сбор, хранение, преобразование, перемещение, выдача)».

Уточним, что такое **фундаментальная** наука и что такое **естественная** наука. К фундаментальным принято относить те науки, основные понятия которых носят общенаучный характер, используются во многих других науках и видах деятельности. Нет, например, сомнений в фундаментальности столь разных наук как математика и философия. В этом же ряду и информатика, так как понятия «информация», «процессы обработки информации» несомненно имеют общенаучную значимость.

Естественные науки - физика, химия, биология и другие - имеют дело с объективными сущностями мира, существующими независимо от нашего сознания. Отнесение к ним информатики отражает единство законов обработки информации в системах самой разной природы - искусственных, биологических, общественных.



Рис. 1.2. К вопросу о месте информатики в системе наук

Однако, многие ученые подчеркивают, что информатика имеет характерные черты и других групп наук - **технических и гуманитарных** (или общественных).

Черты технической науки придают информатике ее аспекты, связанные с созданием и функционированием машинных систем обработки информации. Так, академик А.А.Дородницын определяет состав информатики как «три неразрывно и существенно связанные части: технические средства, программные и алгоритмические». Первоначальное наименование школьного предмета «Основы информатики и вычислительной техники» в настоящее время изменено на «Информатика» (включающее в себя разделы, связанные с изучением технических, программных и алгоритмических средств). Науке информатике присущи и некоторые черты гуманитарной (общественной) науки, что обусловлено ее вкладом в развитие и совершенствование социальной сферы. Таким образом, информатика является комплексной, междисциплинарной отраслью научного знания, как это изображено на рис. 1.2.

1.5. СОЦИАЛЬНЫЕ АСПЕКТЫ ИНФОРМАТИКИ

Термин «социальные аспекты» применительно к большей части наук, тем более фундаментальных, звучит странно. Вряд ли фраза «Социальные аспекты математики» имеет смысл. Однако, информатика - не только наука. Вспомним цитированное выше определение: «... комплекс промышленного, коммерческого, административного и социального воздействия».

И впрямь, мало какие факторы так влияют на социальную сферу обществ (разумеется, находящихся в состоянии относительно спокойного развития, без войн и катаклизмов) как информатизация. Информатизация общества - процесс проникновения информационных технологий во все сферы жизни и деятельности общества. Многие социологи и политологи полагают, что мир стоит на пороге информационного общества. В. А. Извозчиков предлагает следующее определение: «Будем понимать под термином «информационное» («компьютеризированное») общество то, во все сферы жизни и деятельности членов которого включены компьютер, телематика, другие средства информатики в качестве орудий интеллектуального труда, открывающих широкий доступ к сокровищам библиотек, позволяющих с огромной скоростью проводить вычисления и перерабатывать любую информацию, моделировать реальные и прогнозируемые события, процессы, явления, управлять производством, автоматизировать обучение и т.д.». Под «телематикой» понимаются службы обработки информации на расстоянии (кроме традиционных телефона и телеграфа).

Последние полвека информатизация является одной из причин перетока людей из сферы прямого материального производства в, так называемую, информационную сферу. Промышленные рабочие и крестьяне, составлявшие в середине XX века более 2/3 населения, сегодня в развитых странах составляют менее 1/3. Все больше тех, кого называют «белые воротнички» - людей, не создающих материальные ценности непосредственно, а занятых обработкой информации (в самом широком смысле): это и учителя, и банковские служащие, и программисты, и многие другие категории работников. Появились и новые пограничные специальности. Можно ли назвать рабочим программиста, разрабатывающего программы для станков с числовым программным управлением? - По ряду параметров можно, однако его труд не физический, а интеллектуальный.

В табл.1 приведены статистические данные, описывающие изменения в профессиональной структуре труда в США (стране, где информатизация идет особенно быстро) за период с 1970 по 1980 г.

Таблица 1.1 Изменения в структуре труда США за 10 лет

Категория работающих	1970г., %	1980г.,%	Относительный прирост численности, %
Работники сервиса	19,9	21,5	+0,1
Рабочие (промышленные, сельскохозяйственные, фермеры)	38,7	34,2	-11,6
Занятые обработкой информации (всего)	41,5	44,4	+6,7
в том числе:			
менеджеры	8,5	8,7	+2,4
конторские служащие	18,0	18,9	+5,0
специалисты с высшим образованием	15,0	16,8	+12,0

**Таблица 1.2. Профессиональная структура занятости в экономике США
(по данным на 1980 г.)**

Отрасль	Работают с информацией, %	Работают с материальными объектами, %
Обрабатывающая промышленность	40	60
Транспорт и связь	44	56
Оптовая торговля	68	32
Розничная торговля	58	42
Сфера услуг	63	37
Финансовая деятельность	92	8
Государственные учреждения	70	30

Динамика, отраженная в этой таблице, подтверждает сказанное выше. Разумеется, не вся она обусловлена информатизацией, есть и иные факторы, но информатизация вносит решающий вклад. Даже в традиционных сферах деятельности - промышленности и торговле - работа с информацией становится на уровень работы с материальными объектами, в чем убеждают данные, приведенные в табл. 1.2.

За годы, прошедшие с момента публикации этих данных, ситуация изменилась в сторону дальнейшего увеличения доли населения, занятого в профессиональном труде обработкой информации. К середине 90-х годов численность «информационных работников» (к которым причисляют всех, в чьей профессиональной деятельности доминирует умственный труд) достигла в США 60%. Добавим, что за те же годы производительность труда в США за счет научно-технического прогресса (ведь информатизация - его главная движущая сила) в целом выросла на 37%.

Информатизация сильнейшим образом влияет на структуру экономики ведущих в экономическом отношении стран. В числе их лидирующих отраслей промышленности традиционные добывающие и обрабатывающие отрасли оттеснены максимально наукоемкими производствами электроники, средств связи и вычислительной техники (так называемой, сферой высоких технологий). В этих странах постоянно растут капиталовложения в научные исследования, включая фундаментальные науки. Темпы развития сферы высоких технологий и уровень прибылей в ней превышают в 5-10 раз темпы развития традиционных отраслей производства. Такая политика имеет и социальные последствия - увеличение потребности в высокообразованных специалистах и связанный с этим прогресс системы высшего образования. Информатизация меняет и облик традиционных отраслей промышленности и сельского хозяйства. Промышленные роботы, управляемые ЭВМ, станки с ЧПУ стали обычным оборудованием. Новейшие технологии в сельскохозяйственном производстве не только увеличивают производительность труда, но и облегчают его, вовлекают более образованных людей.

Казалось бы, компьютеризация и информационные технологии несут в мир лишь благодать, но социальная сфера столь сложна, что последствия любого, даже гораздо менее глобального процесса, редко бывают однозначными. Рассмотрим, например, такие социальные последствия информатизации как рост производительности труда, интенсификацию труда, изменение условий труда. Все это, с одной стороны, улучшает условия жизни многих людей, повышает степень материального и интеллектуального комфорта, стимулирует рост числа высокообразованных людей, а с другой - является источником повышенной социальной напряженности. Например, появление на производстве промышленных роботов ведет к полному изменению технологии, которая перестает быть ориентированной на человека. Тем самым меняется номенклатура профессий. Значительная часть людей вынуждена менять либо специальность, либо место работы - рост миграции населения характерен для большинства развитых стран. Государство и частные фирмы поддерживают систему повышения квалификации и переподготовки, но не все люди справляются с сопутствующим стрессом. Прогрессом информатики порожден и другой достаточно опасный для демократического общества процесс - все большее количество данных о каждом гражданине сосредоточивается в разных (государственных и негосударственных) банках данных. Это и данные о профессиональной карьере (базы данных отделов кадров), здоровье (базы данных учреждений здравоохранения), имущественных возможностях (базы данных страховых компаний), перемещении по миру и т.д. (не говоря уже о тех, которые копят специальные службы). В каждом кон-

кретном случае создание банка может быть оправдано, но в результате возникает система невиданной раньше ни в одном тоталитарном обществе прозрачности личности, чреватой возможным вмешательством государства или злоумышленников в частную жизнь. Одним словом, жизнь в «информационном обществе» легче, по-видимому, не становится, а вот то, что она значительно меняется - несомненно.

Трудно, живя в самом разгаре описанных выше процессов, взвесить, чего в них больше - положительного или отрицательного, да и четких критериев для этого не существует. Тяжелая физическая работа в не слишком комфортабельных условиях, но с уверенностью, что она будет постоянным источником существования для тебя и твоей семьи, с одной стороны, или интеллектуальный труд в комфортабельном офисе, но без уверенности в завтрашнем дне. Что лучше? Конечно, вряд ли стоит уподобляться английским рабочим, ломавшим в конце XVIII века станки, лишавшие их работы, но правительство и общество обязаны помнить об отрицательных социальных последствиях информатизации и научно-технического прогресса в целом и искать компенсационные механизмы.

1.6. ПРАВОВЫЕ АСПЕКТЫ ИНФОРМАТИКИ

Деятельность программистов и других специалистов, работающих в сфере информатики, все чаще выступает в качестве объекта правового регулирования. Некоторые действия при этом могут быть квалифицированы как правонарушения (преступления).

Правовое сознание в целом, а в области информатики особенно, в нашем обществе находится на низком уровне. Все ли знают ответы на следующие вопросы:

- можно ли, не копируя купленную программу, предоставить возможность пользоваться ею другому лицу;
- кому принадлежит авторское право на программу, созданную студентом в ходе выполнения дипломной работы;
- можно ли скопировать купленную программу для себя самого, чтобы иметь резервную копию;
- можно ли декомпилировать программу, чтобы разобраться в ее деталях или исправить ошибки;
- в чем состоит разница между авторским и имущественным правом.

Вопросов, подобных этим, возникает множество. Есть, конечно, такие, ответы на которые очевидны: нельзя создавать вирусы, нельзя хулиганить в сетях, нельзя в некоммерческих телеконференциях запускать коммерческую информацию, нельзя вскрывать и искажать защищенную информацию в чужих базах данных и т.д., т.е. совершать поступки, которые могут быть объектом уголовного преследования. Но на многие вопросы ответы отнюдь не очевидны, а иногда казуистически запутаны, причем не только в нашей стране. Остановимся на правовом регулировании в области информатики в России более подробно.

Необходимо отметить, что регулирование в сфере, связанной с защитой информации, программированием и т.д., является для российского законодательства принципиально новым, еще слабо разработанным направлением. К 1992 году был принят Закон Российской Федерации «О ПРАВОВОЙ ОХРАНЕ ПРОГРАММ ДЛЯ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН И БАЗ ДАННЫХ», содержащий обширный план приведения российского законодательства в сфере информатики в соответствие с мировой практикой. Действие этого Закона распространяется на отношения, связанные с созданием и использованием программ для ЭВМ и баз данных. Также предусматривалось внести изменения и дополнения в Гражданский кодекс РФ, в Уголовный кодекс РФ, другие законодательные акты, связанные с вопросами правовой охраны программ для электронных вычислительных машин и баз данных, привести решения Правительства РФ в соответствие с Законом, обеспечить пересмотр и отмену государственными ведомствами и другими организациями РФ их нормативных актов, противоречащих указанному Закону, обеспечить принятие нормативных актов в соответствии с указанным Законом и т.д.

Главное содержание данного Закона - юридическое определение понятий, связанных с авторством и распространением компьютерных программ и баз данных, таких как Авторство, Адаптация, База данных, Воспроизведение, Декомпилирование. Использование, Модификация и т.д., а также установление прав, возникающих при создании программ и баз данных - авторских, имущественных, на передачу, защиту, регистрацию, неприкосновенность и т.д.

Авторское право распространяется на любые программы для ЭВМ и базы данных (как вы-

пущенные, так и не выпущенные в свет), представленные в объективной форме, независимо от их материального носителя, назначения и достоинства. Авторское право распространяется на программы для ЭВМ и базы данных, являющиеся результатом творческой деятельности автора. Творческий характер деятельности автора предполагается до тех пор, пока не доказано обратное.

Предоставляемая настоящим Законом правовая охрана распространяется на все виды программ для ЭВМ (в том числе на операционные системы и программные комплексы), которые могут быть выражены на любом языке и в любой форме, и на базы данных, представляющие собой результат творческого труда по подбору и организации данных. Предоставляемая правовая охрана не распространяется на идеи и принципы, лежащие в основе программы для ЭВМ и базы данных или какого-либо их элемента, в том числе идеи и принципы организации интерфейса и алгоритма, а также языки программирования.

Авторское право на программы для ЭВМ и базы данных возникает в силу их создания. Для признания у. осуществления авторского права на программы для ЭВМ и базы данных не требуется опубликования, регистрации или соблюдения иных формальностей. Авторское право на базу данных признается при условии соблюдения авторского права на каждое из произведений, включенных в базу данных.

Автором программы для ЭВМ и базы данных признается физическое лицо, в результате творческой деятельности которого они созданы.

Если программа для ЭВМ и база данных созданы совместной творческой деятельностью двух и более физических лиц, то, независимо от того, состоит ли программа для ЭВМ или база данных из частей, каждая из которых имеет самостоятельное значение, или является неделимой, каждое из этих лиц признается автором такой программы для ЭВМ и базы данных.

Автору программы для ЭВМ или базы данных или иному правообладателю принадлежит исключительное право осуществлять и (или) разрешать осуществление следующих действий:

- выпуск в свет программы для ЭВМ и базы данных;
- воспроизведение программы для ЭВМ и базы данных (полное или частичное) в любой форме, любыми способами;
- распространение программы для ЭВМ и баз данных;
- модификацию программы для ЭВМ и базы данных, в том числе перевод программы для ЭВМ и базы данных с одного языка на другой;
- иное использование программы для ЭВМ и базы данных.

Однако, **имущественные права** на программы для ЭВМ и базы данных, созданные в порядке выполнения служебных обязанностей или по заданию работодателя, принадлежат работодателю, если в договоре между ним и автором не предусмотрено иное. Таким образом, имущественное право на программу, созданную в ходе дипломного проектирования, принадлежит не автору, а вузу - по крайней мере, пока между ними не будет заключено специальное соглашение.

Имущественные права на программу для ЭВМ и базу данных могут быть переданы полностью или частично другим физическим или юридическим лицам по договору. Договор заключается в письменной форме и должен устанавливать следующие существенные условия: объем и способы использования программы для ЭВМ или базы данных, порядок выплаты и размер вознаграждения, срок действия договора.

Лицо, правомерно владеющее экземпляром программы для ЭВМ или базы данных, вправе без получения дополнительного разрешения правообладателя осуществлять любые действия, связанные с функционированием программы для ЭВМ или базы данных в соответствии с их назначением, в том числе запись и хранение в памяти ЭВМ, а также исправление явных ошибок. Запись и хранение в памяти ЭВМ допускаются в отношении одной ЭВМ или одного пользователя в сети, если иное не предусмотрено договором с правообладателем. Также допускается без согласия правообладателя и без выплаты ему дополнительного вознаграждения осуществлять следующие действия:

- адаптацию программы для ЭВМ или базы данных;
- изготавливать или поручать изготовление копии программы для ЭВМ или базы данных при условии, что эта копия предназначена только для архивных целей и при необходимости (в случае, когда оригинал программы для ЭВМ или базы данных утерян, уничтожен или стал непригодным для использования) для замены правомерно приобретенного экземпляра.

Лицо, правомерно владеющее экземпляром программы для ЭВМ, вправе без согласия пра-

вообладателя и без выплаты дополнительного вознаграждения выполнять **декомпилирование** программы для ЭВМ с тем, чтобы изучить кодирование и структуру этой программы при следующих условиях:

- информация, необходимая для взаимодействия независимо разработанной данным лицом программы для ЭВМ с другими программами, недоступна из других источников;

- информация, полученная в результате этого декомпилирования, может использоваться лишь для организации взаимодействия независимо разработанной данным лицом программы для ЭВМ с другими программами, а не для составления новой программы для ЭВМ, по своему виду существенно схожей с декомпилируемой программой.

Свободная **перепродажа** экземпляра программы для ЭВМ и базы данных допускается без согласия правообладателя и без выплаты ему дополнительного вознаграждения после первой продажи или другой передачи права собственности на этот экземпляр.

Выпуск под своим именем чужой программы для ЭВМ или базы данных, а также незаконное воспроизведение или распространение таких произведений влечет за собой уголовную ответственность.

В настоящее время уголовное законодательство РФ не в полной мере учитывает все возможные компьютерные преступления. Вообще же, в законодательной практике многих стран отмечены различные виды компьютерных преступлений и разработаны методы борьбы с ними.

Компьютерные преступления условно можно разделить на две большие категории:

- 1) преступления, связанные с вмешательством в работу компьютеров;
- 2) преступления, использующие компьютеры как необходимые технические средства.

Можно выделить следующие виды компьютерной преступности 1-го вида:

- несанкционированный доступ в компьютерные сети и системы, банки данных с целью шпионажа или диверсии (военного, промышленного, экономического), с целью, так называемого, компьютерного хищения или из хулиганских побуждений;

- ввод в программное обеспечение, так называемых, «логических бомб», срабатывающих при определенных условиях (логические бомбы, угрожающие уничтожением данных, могут использоваться для шантажа владельцев информационных систем или выполнять новые, не планировавшиеся владельцем программы, функции при сохранении работоспособности системы; известны случаи, когда программисты вводили в программы финансового учета команды, переводящие на счета этих программистов денежные суммы или скрывающие денежные суммы от учета, что позволяло незаконно получать их);

- разработку и распространение компьютерных вирусов;
- преступную небрежность в разработке, изготовлении и эксплуатации программно-вычислительных комплексов, приведшую к тяжким последствиям;

- подделку компьютерной информации (продукции) и сдачу заказчикам неработоспособных программ, подделку результатов выборов, референдумов;

- хищение компьютерной информации (нарушение авторского права и права владения программными средствами и базами данных).

Среди компьютерных преступлений 2-го вида, т.е. использующих компьютер как средство преступления, следует отметить преступления, спланированные на основе компьютерных моделей, например, в сфере бухгалтерского учета.

Для современного состояния правового регулирования сферы, связанной с информатикой, в России в настоящее время наиболее актуальными являются вопросы, связанные с нарушением авторских прав. Большая часть программного обеспечения, использующегося отдельными программистами и пользователями и целыми организациями, приобретена в результате незаконного копирования, т.е. хищения. Назрела потребность узаконить способы борьбы с этой порочной практикой, поскольку она мешает, прежде всего, развитию самой информатики.

1.7. ЭТИЧЕСКИЕ АСПЕКТЫ ИНФОРМАТИКИ

Далеко не все правила, регламентирующие деятельность в сфере информатики, можно свести в правовым нормам. Очень многое определяется соблюдением неписаных правил поведения для тех, кто причастен к миру компьютеров. Впрочем, в этом отношении информатика ничуть не отличается от любой другой сферы деятельности человека в обществе.

Как и в любой другой большой и разветвленной сфере человеческой деятельности, в информатике к настоящему времени сложились определенные морально-этические нормы поведения и деятельности.

Морально-этические нормы в среде информатиков отличаются от этики повседневной жизни несколько большей открытостью, альтруизмом. Большинство нынешних специалистов-информатиков сформировались и приобрели свои знания и квалификацию благодаря бескорыстным консультациям и содействию других специалистов. Очевидно, поэтому они готовы оказать бескорыстную помощь, дать совет или консультацию, предоставить компьютер для выполнения каких-либо манипуляций с дискетами и т.д. Ярким примером особой психологической атмосферы в среде информатиков является расширяющееся международное движение программистов, предоставляющих созданные ими программные средства для свободного распространения.

Это - положительные аспекты, но есть и отрицательные. Обратим внимание на язык информатиков. Сленг российских информатиков построен на искаженных под русское произношение англоязычных терминах и аббревиатурах, введенных иностранными фирмами - разработчиками компьютеров и программного обеспечения в технической документации. Одновременно формируется и набор сленговых слов, заимствованных из русского языка на основе аналогий и ассоциаций по сходству и смежности (например: архивированный - «утопанный», компьютер - «железо» или «тачка» и т.д.). С тем, что многие специальные термины пришли к нам из США, приходится мириться. Никто сегодня уже не перейдет от термина «принтер» к аналогичному «автоматическое цифровое печатающее устройство» (которым пользовались не так уж давно). Приживаемости подобных слов в отечественной литературе способствует, в частности, их относительная краткость. Однако трудно понять, зачем в телеконференции учителя иногда именуют себя «тичерами» - от этого они лучше не становятся. Итак, одно из этических правил - не искажай родной язык.

Особую остроту этические проблемы приобретают при работе в глобальных телекоммуникационных сетях. Вскрыть защиту чужой базы данных – уголовное преступление. А можно ли позволять себе нецензурные выражения или прозрачные их эвфемизмы? Коммерческую рекламу в некоммерческой телеконференции? Независимо от того, предусмотрено за это законом возмездие или нет, порядочный человек этого делать не станет.

Этика - система норм нравственного поведения человека. Порядочный человек не прочтет содержимое дискеты, забытой соседом на рабочем месте, не потому, что это грозит ему наказанием, а потому, что это безнравственный поступок; не скопирует программу в отсутствие ее хозяина не потому, что на него могут подать в суд, а потому, что этот поступок осудят его коллеги. Всякий раз, собираясь совершить сомнительный поступок в сфере профессиональной деятельности, человек должен задуматься, соответствует ли он этическим нормам, сложившимся в профессиональном сообществе.

Контрольные вопросы

1. Что общего и в чем различие информатики и кибернетики?
2. Какие определения информатики вы знаете?
3. Какова общая структура современной информатики?
4. Какие существуют наиболее известные информационные технологии?
5. Какое место занимает информатика в системе наук?
6. Что принято понимать под «информационным обществом»?
7. Каковы основные социальные последствия информатизации общества?
8. Какими нормативными актами регулируются отношения в сфере информатики?
9. В чем состоит авторское право на программные средства и базы данных?
10. В чем состоит имущественное право на программные средства и базы данных?
11. Как осуществляется защита авторских и имущественных прав?
12. Охарактеризуйте виды компьютерных преступлений.
13. Какие этические проблемы существуют, по вашему мнению, в современной информатике?

§ 2. ИНФОРМАЦИЯ, ЕЕ ВИДЫ И СВОЙСТВА

2.1. РАЗЛИЧНЫЕ УРОВНИ ПРЕДСТАВЛЕНИЙ ОБ ИНФОРМАЦИИ

Ранее мы неоднократно употребляли термин «информация», никак его при этом не раскрывая.

Понятие **информация** является одним из фундаментальных в современной науке вообще и базовым для изучаемой нами информатики. Информацию наряду с веществом и энергией рассматривают в качестве важнейшей сущности мира, в котором мы живем. Однако, если задаться целью формально определить понятие «информация», то сделать это будет чрезвычайно сложно. Аналогичными «неопределяемыми» понятиями, например, в математике является «точка» или «прямая». Так, можно сделать некоторые утверждения, связанные с этими математическими понятиями, но сами они не могут быть определены с помощью более элементарных понятий.

В простейшем бытовом понимании с термином «информация» обычно ассоциируются некоторые сведения, данные, знания и т.п. Информация передается в виде **сообщений**, определяющих форму и представление передаваемой информации. Примерами сообщений являются музыкальное произведение; телепередача; команды регулировщика на перекрестке; текст, распечатанный на принтере; данные, полученные в результате работы составленной вами программы и т.д. При этом предполагается, что имеются «источник информации» и «получатель информации».

Сообщение от источника к получателю передается посредством какой-нибудь среды, являющейся в таком случае «каналом связи» (рис. 1.3). Так, при передаче речевого сообщения в качестве такого канала связи можно рассматривать воздух, в котором распространяются звуковые волны, а в случае передачи письменного сообщения (например, текста, распечатанного на принтере) каналом сообщения можно считать лист бумаги, на котором напечатан текст.

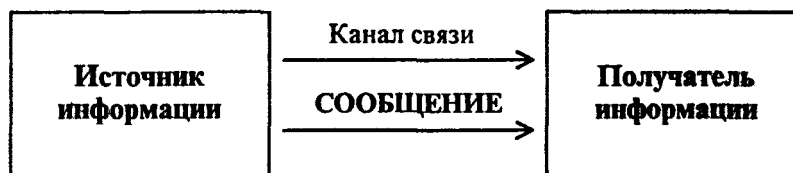


Рис. 1.3. Схема передачи информации

Человеку свойственно субъективное восприятие информации через некоторый набор ее свойств: важность, достоверность, своевременность, доступность и т.д. В этом смысле одно и то же сообщение, передаваемое от источника к получателю, может передавать информацию в разной степени. Так, например, вы хотите сообщить о неисправности компьютера. Для инженера из группы технического обслуживания сообщение «компьютер сломался» явно содержит больше информации, чем для вахтера. Но, в свою очередь, для инженера сообщение «не включается дисплей» содержит информации больше, чем первое, поскольку в большей степени снимает неопределенность, связанную с причиной неисправности компьютера. Как видно, одно и то же сообщение для различных пользователей несет различную информацию.

Использование терминов «больше информации» или «меньше информации» подразумевает некую возможность ее **измерения** (или хотя бы количественного соотнесения). При субъективном восприятии измерение информации возможно лишь в виде установления некоторой порядковой шкалы для оценки «больше» - «меньше», да и то субъективной, поскольку на свете немало людей, для которых, например, оба сообщения, использованных выше в качестве примера, вообще не несут никакой информации. Такое становится невозможным при введении объективных характеристик, из которых для информации важнейшей является количество. Однако, при объективном измерении количества информации следует заведомо отрешиться от восприятия ее с точки зрения субъективных свойств, примеры которых перечислены выше. Более того, не исключено, что не всякая информация будет иметь объективно измеряемое количество - все зависит от того, как будут введены единицы измерения. Не исключено и то, что при разных способах введения единиц измерения информация, содержащаяся в двух допускающих измерение сообщениях, будет по разному соотноситься.

2.2. НЕПРЕРЫВНАЯ И ДИСКРЕТНАЯ ИНФОРМАЦИЯ

Чтобы сообщение было передано от источника к получателю, необходима некоторая материальная субстанция - **носитель** информации. Сообщение, передаваемое с помощью носителя, назовем сигналом. В общем случае **сигнал** - это изменяющийся во времени физический процесс. Такой процесс может содержать различные характеристики (например, при передаче электрических сигналов могут изменяться напряжение и сила тока). Та из характеристик, которая используется для представления сообщений, называется **параметром сигнала**.

В случае когда параметр сигнала принимает последовательное во времени конечное число значений (при этом все они могут быть пронумерованы), сигнал называется **дискретным**, а сообщение, передаваемое с помощью таких сигналов - дискретным сообщением. Информация, передаваемая источником, в этом случае также называется дискретной. Если же источник вырабатывает непрерывное сообщение (соответственно параметр сигнала - непрерывная функция от времени), соответствующая информация называется непрерывной. Пример дискретного сообщения - процесс чтения книги, информация в которой представлена текстом, т.е. дискретной последовательностью отдельных значков (букв). Примером непрерывного сообщения служит человеческая речь, передаваемая модулированной звуковой волной; параметром сигнала в этом случае является давление, создаваемое этой волной в точке нахождения приемника - человеческого уха.

Непрерывное сообщение может быть представлено непрерывной функцией, заданной на некотором отрезке $[a, b]$ (см. рис. 1.4). Непрерывное сообщение можно преобразовать в дискретное (такая процедура называется **дискретизацией**). Для этого из бесконечного множества значений этой функции (параметра сигнала) выбирается их определенное число, которое приближенно может характеризовать остальные значения. Один из способов такого выбора состоит в следующем. Область определения функции разбивается точками $x_1, x_2, \dots, x_n$, на отрезки равной длины и на каждом из этих отрезков значение функции принимается постоянным и равным, например, среднему значению на этом отрезке; полученная на этом этапе функция называется в математике ступенчатой. Следующий шаг - проецирование значений «ступенек» на ось значений функции (ось ординат). Полученная таким образом последовательность значений функции $y_1, y_2, \dots, y_n$ является дискретным представлением непрерывной функции, точность которого можно неограниченно улучшать путем уменьшения длин отрезков разбиения области значений аргумента.

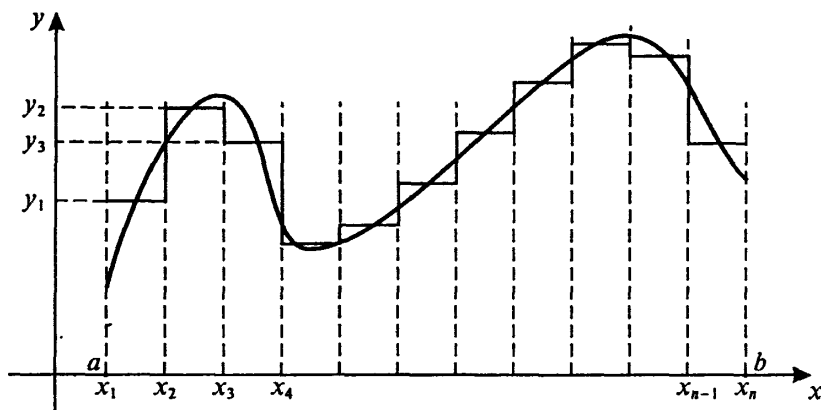


Рис. 1.4. Процедура дискретизации непрерывного сообщения

Ось значений функции можно разбить на отрезки с заданным шагом и отобразить каждый из выделенных отрезков из области определения функции в соответствующий отрезок из множества значений (рис. 1.4). В итоге получим конечное множество чисел, определяемых, например, по середине или одной из границ таких отрезков.

Таким образом, любое сообщение может быть представлено как дискретное, иначе говоря последовательностью знаков некоторого алфавита.

Возможность дискретизации непрерывного сигнала с любой желаемой точностью (для возрастания точности достаточно уменьшить шаг) принципиально важна с точки зрения информатики. Компьютер - цифровая машина, т. е. внутреннее представление информации в нем дискретно. Дискретизация входной информации (если она непрерывна) позволяет сделать ее пригодной для компьютерной обработки.

Существуют и другие вычислительные машины - аналоговые ЭВМ. Они используются обычно для решения задач специального характера и широкой публике практически не известны. Эти ЭВМ в принципе не нуждаются в дискретизации входной информации, так как ее внутреннее

представление у них непрерывно. В этом случае все наоборот - если внешняя информация дискретна, то ее «перед употреблением» необходимо преобразовать в непрерывную.

2.3. ЕДИНИЦЫ КОЛИЧЕСТВА ИНФОРМАЦИИ: ВЕРОЯТНОСТНЫЙ И ОБЪЕМНЫЙ ПОДХОДЫ

Определить понятие «количество информации» довольно сложно. В решении этой проблемы существуют два основных подхода. Исторически они возникли почти одновременно. В конце 40-х годов XX века один из основоположников кибернетики американский математик Клод Шеннон развил вероятностный подход к измерению количества информации, а работы по созданию ЭВМ привели к «объемному» подходу.

Вероятностный подход

Рассмотрим в качестве примера опыт, связанный с бросанием правильной игральной кости, имеющей N граней (наиболее распространенным является случай шестигранной кости: $N = 6$). Результаты данного опыта могут быть следующие: выпадение грани с одним из следующих знаков: $1, 2, \dots, N$.

Введем в рассмотрение численную величину, измеряющую неопределенность - **энтропию** (обозначим ее H). Величины N и H связаны между собой некоторой функциональной зависимостью:

$$H = f(N), \quad (1.1)$$

а сама функция f является возрастающей, неотрицательной и определенной (в рассматриваемом нами примере) для $N = 1, 2, \dots, 6$.

Рассмотрим процедуру бросания кости более подробно:

1) готовимся бросить кость; исход опыта неизвестен, т.е. имеется некоторая неопределенность; обозначим ее H_1 ;

2) кость брошена; информация об исходе данного опыта получена; обозначим количество этой информации через I ;

3) обозначим неопределенность данного опыта после его осуществления через H_2 . За количество информации, которое получено в ходе осуществления опыта, примем разность неопределенностей «до» и «после» опыта:

$$I = H_1 - H_2 \quad (1.2)$$

Очевидно, что в случае, когда получен конкретный результат, имевшаяся неопределенность снята ($H_2 = 0$), и, таким образом, количество полученной информации совпадает с первоначальной энтропией. Иначе говоря, неопределенность, заключенная в опыте, совпадает с информацией об исходе этого опыта. Заметим, что значение H_2 могло быть и не равным нулю, например, в случае, когда в ходе опыта следующей выпала грань со значением, большим «3».

Следующим важным моментом является определение вида функции f в формуле (1.1). Если варьировать число граней N и число бросаний кости (обозначим эту величину через M), общее число исходов (векторов длины M , состоящих из знаков $1, 2, \dots, N$) будет равно N в степени M :

$$X = N^M. \quad (1.3)$$

Так, в случае двух бросаний кости с шестью гранями имеем: $X = 6^2 = 36$. Фактически каждый исход X есть некоторая пара (X_1, X_2) , где X_1 и X_2 - соответственно исходы первого и второго бросаний (общее число таких пар - X).

Ситуацию с бросанием M раз кости можно рассматривать как некую сложную систему, состоящую из независимых друг от друга подсистем - «однократных бросаний кости». Энтропия такой системы в M раз больше, чем энтропия одной системы (так называемый «принцип аддитивности энтропии»):

$$f(6^M) = M \cdot f(6)$$

Данную формулу можно распространить и на случай любого N :

$$F(N^M) = M \cdot f(N) \quad (1.4)$$

Прологарифмируем левую и правую части формулы (1.3): $\ln X = M \cdot \ln N$, $M = \ln X / \ln N$. Подставляем полученное для M значение в формулу (1.4):

$$f(x) = \frac{\ln X}{\ln N} \cdot f(N).$$

Обозначив через K положительную константу, получим: $f(X) = K \cdot \ln X$, или, с учетом (1.1), $H = K \cdot \ln N$. Обычно принимают $K = 1 / \ln 2$. Таким образом

$$H = \log_2 N. \quad (1.5)$$

Это - **формула Хартли**.

Важным при введении какой-либо величины является вопрос о том, что принимать за единицу ее измерения. Очевидно, H будет равно единице при $N = 2$. Иначе говоря, в качестве единицы принимается количество информации, связанное с проведением опыта, состоящего в получении одного из двух равновероятных исходов (примером такого опыта может служить бросание монеты при котором возможны два исхода: «орел», «решка»). Такая единица количества информации называется «бит».

Все N исходов рассмотренного выше опыта являются равновероятными и поэтому можно считать, что на «долю» каждого исхода приходится одна N -я часть общей неопределенности опыта: $(\log_2 N)1N$. При этом вероятность i -го исхода P_i равняется, очевидно, $1/N$.

Таким образом,

$$H = \sum_{i=1}^N P_i \cdot \log_2 \left(\frac{1}{P_i} \right).$$

Та же формула (1.6) принимается за меру энтропии в случае, когда вероятности различных исходов опыта **неравновероятны** (т.е. P_i могут быть различны). Формула (1.6) называется **формулой Шеннона**.

В качестве примера определим количество информации, связанное с появлением каждого символа в сообщениях, записанных на русском языке. Будем считать, что русский алфавит состоит из 33 букв и знака «пробел» для разделения слов. По формуле (1.5)

$$H = \log_2 34 \approx 5 \text{ бит.}$$

Однако, в словах русского языка (равно как и в словах других языков) различные буквы встречаются неодинаково часто. Ниже приведена табл. 1.3 вероятностей частоты употребления различных знаков русского алфавита, полученная на основе анализа очень больших по объему текстов.

Воспользуемся для подсчета H формулой (1.6); $H \approx 4,72$ бит. Полученное значение H , как и можно было предположить, меньше вычисленного ранее. Величина H , вычисляемая по формуле (1.5), является максимальным количеством информации, которое могло бы приходиться на один знак.

Таблица 1.3. Частотность букв русского языка

i	Символ	P(i)	i	Символ	P(i)	i	Символ	P(i)
1	Пробел	0,175	13		0,028	24	Г	0.012

2	О	0,090	14	М	0,026	25	Ч	0,012
3	Е	0,072	15	Д	0,025	26	И	0,010
4	Ё	0,072	16	П	0,023	27	Х	0,009
5	А	0,062	17	У	0,021	28	Ж	0,007
6	И	0,062	18	Я	0,018	29	Ю	0,006
7	Т	0,053	19	Ы	0,016	30	Ш	0,006
8	Н	0,053	20	З	0,016	31	Ц	0,004
9	С	0,045	21	Ь	0,014	32	Щ	0,003
10	Р	0,040	22	Ъ	0,014	33	Э	0,003
11	В	0,038	23	Б	0,014	34	Ф	0,002
12	Л	0,035						

Аналогичные подсчеты H можно провести и для других языков, например, использующих латинский алфавит - английского, немецкого, французского и др. (26 различных букв и «пробел»). По формуле (1.5) получим

$$H = \log_2 27 \approx 4,76 \text{ бит.}$$

Как и в случае русского языка, частота появления тех или иных знаков не одинакова.

Если расположить все буквы данных языков в порядке убывания вероятностей, то получим следующие последовательности:

АНГЛИЙСКИЙ ЯЗЫК: «пробел», Е, Т, А, О, N, R, ...

НЕМЕЦКИЙ ЯЗЫК: «пробел», Е, N, I, S, T, R, ...

ФРАНЦУЗСКИЙ ЯЗЫК: «пробел», Е, S, A, N, I, T, ...

Рассмотрим алфавит, состоящий из двух знаков 0 и 1. Если считать, что со знаками 0 и 1 в двоичном алфавите связаны одинаковые вероятности их появления ($P(0) = P(1) = 0,5$), то количество информации на один знак при двоичном кодировании будет равно

$$H = \log_2 2 = 1 \text{ бит.}$$

Таким образом, количество информации (в битах), заключенное в двоичном слове, равно числу двоичных знаков в нем.

Объемный подход

В двоичной системе счисления знаки 0 и 1 будем называть **битами** (от английского выражения Binary digiTs - двоичные цифры). Отметим, что создатели компьютеров отдают предпочтение именно двоичной системе счисления потому, что в техническом устройстве наиболее просто реализовать два противоположных физических состояния: некоторый физический элемент, имеющий два различных состояния: намагниченность в двух противоположных направлениях; прибор, пропускающий или нет электрический ток; конденсатор, заряженный или незаряженный и т.п. В компьютере бит является наименьшей возможной единицей информации. Объем информации, записанной двоичными знаками в памяти компьютера или на внешнем носителе информации подсчитывается просто по количеству требуемых для такой записи двоичных символов. При этом, в частности, невозможно нецелое число битов (в отличие от вероятностного подхода).

Для удобства использования введены и более крупные, чем бит, единицы количества информации. Так, двоичное слово из восьми знаков содержит один, **байт информации**, 1024 байта образуют **килобайт** (кбайт), 1024 килобайта - **мегабайт** (Мбайт), а 1024 мегабайта - **гигабайт** (Гбайт).

Между вероятностным и объемным количеством информации соотношение неоднозначное. Далеко не всякий текст, записанный двоичными символами, допускает измерение объема информации в кибернетическом смысле, но заведомо допускает его в объемном. Далее, если некоторое сообщение допускает измеримость количества информации в обоих смыслах, то они не обязательно совпадают, при этом кибернетическое количество информации не может быть больше объемного.

В дальнейшем тексте данного учебника практически всегда количество информации понимается в объемном смысле.

2.4. ИНФОРМАЦИЯ: БОЛЕЕ ШИРОКИЙ ВЗГЛЯД

Как ни важно измерение информации, нельзя сводить к нему все связанные с этим понятием проблемы. При анализе информации социального (в широком смысле) происхождения на первый план могут выступить такие ее свойства, как истинность, своевременность, ценность, полнота и т.д. Их невозможно оценить в терминах «уменьшение неопределенности» (вероятностный подход) или числа символов (объемный подход). Обращение к качественной стороне информации породило иные подходы к ее оценке. При **аксиологическом** подходе стремятся исходить из ценности, практической значимости информации, т.е. качественных характеристик, значимых в социальной системе. При **семантическом** подходе информация рассматривается с точки зрения как формы, так и содержания. При этом информацию связывают с **тезаурусом**, т.е. полнотой систематизированного набора данных о предмете информации. Отметим, что эти подходы не исключают количественного анализа, но он становится существенно сложнее и должен базироваться на современных методах математической статистики.

Понятие информации нельзя считать лишь техническим, междисциплинарным и даже наддисциплинарным термином. Информация - это фундаментальная философская категория. Дискуссии ученых о философских аспектах информации надежно показали несводимость информации ни к одной из этих категорий. Концепции и толкования, возникающие на пути догматических подходов, оказываются слишком частными, односторонними, не охватывающими всего объема этого понятия.

Попытки рассмотреть категорию информации с позиций основного вопроса философии привели к возникновению двух противостоящих концепций - так называемых, **функциональной и атрибутивной**. «Атрибутисты» квалифицируют информацию как свойство всех материальных объектов, т.е. как атрибут материи. «функционалисты» связывают информацию лишь с функционированием сложных, самоорганизующихся систем. Оба подхода, скорее всего, неполны. Дело в том, что природа сознания, духа по сути своей является информационной, т.е. создание суть менее общее понятие по отношению к категории «информация». Нельзя признать корректными попытки сведения более общего понятия к менее общему. Таким образом, информация и информационные процессы, если иметь в виду решение основного вопроса философии, опосредуют материальное и духовное, т.е. вместо классической постановки этого вопроса получается два новых: о соотношении материи и информации и о соотношении информации и сознания (духа);

Можно попытаться дать философское определение информации с помощью указания на связь определяемого понятия с категориями **отражения и активности**. Информация есть содержание образа, формируемого в процессе отражения. Активность входит в это определение в виде представления о формировании некоего образа в процессе отражения некоторого субъект-объектного отношения. При этом не требуется указания на связь информации с материей, поскольку как субъект, так и объект процесса отражения могут принадлежать как к материальной, так и к духовной сфере социальной жизни. Однако существенно подчеркнуть, что материалистическое решение основного вопроса философии требует признания необходимости существования материальной среды - носителя информации в процессе такого отражения. Итак, информацию следует трактовать как имманентный (неотъемлемо присущий) атрибут материи, необходимый момент ее самодвижения и саморазвития. Эта категория приобретает особое значение применительно к высшим формам движения материи - биологической и социальной.

Данное выше определение схватывает важнейшие характеристики информации. Оно не противоречит тем знаниям, которые накоплены по этой проблематике, а наоборот, является выражением наиболее значимых.

Современная практика психологии, социологии, информатики диктует необходимость перехода к информационной трактовке сознания. Такая трактовка оказывается чрезвычайно плодотворной и позволяет, например, рассмотреть с общих позиций индивидуальное и общественное сознание. Генетически индивидуальное и общественное сознание неразрывны и в то же время общественное сознание не есть простая сумма индивидуальных, поскольку оно включает информационные потоки и процессы между индивидуальными сознаниями.

В социальном плане человеческая деятельность предстает как взаимодействие реальных человеческих коммуникаций с предметами материального мира. Поступившая извне к человеку информация является отпечатком, снимком сущностных сил природы или другого человека. Таким образом, с единых методологических позиций может быть рассмотрена деятельность индивидуального и общественного сознания, экономическая, политическая, образовательная деятельность различных субъектов социальной системы.

Данное выше определение информации как философской категории не только затрагивает физические аспекты существования информации, но и фиксирует ее социальную значимость.

Одной из важнейших черт функционирования современного общества выступает его информационная оснащенность. В ходе своего развития человеческое общество прошло через пять информационных революций. Первая из них была связана с введением языка, вторая - письменности, третья - книгопечатания, четвертая - телесвязи, и, наконец, пятая - компьютеров (а также магнитных и оптических носителей хранения информации). Каждый раз новые информационные технологии поднимали информированность общества на несколько порядков, радикально меняя объем и глубину знания, а вместе с этим и уровень культуры в целом.

Одна из целей философского анализа понятия информации - указать место информационных технологий в развитии форм движения материи, в прогрессе человечества и, в том числе, в развитии разума как высшей отражательной способности материи. На протяжении десятков тысяч лет сфера разума развивалась исключительно через общественную форму сознания. С появлением компьютеров начались разработки систем искусственного интеллекта, идущих по пути моделирования общих интеллектуальных функций индивидуального сознания.

2.5. ИНФОРМАЦИЯ И ФИЗИЧЕСКИЙ МИР

Известно большое количество работ, посвященных физической трактовке информации. Эти работы в значительной мере построены на основе аналогии формулы Больцмана, описывающей энтропию статистической системы материальных частиц, и формулы Хартли.

Заметим, что при всех выводах формулы Больцмана явно или неявно предполагается, что макроскопическое состояние системы, к которому относится функция энтропии, реализуется на микроскопическом уровне как сочетание механических состояний очень большого числа частиц, образующих систему (молекул). Задачи же кодирования и передачи информации, для решения которых Хартли и Шенноном была разработана вероятностная мера информации, имели в виду очень узкое техническое понимание информации, почти не имеющее отношения к полному объему этого понятия. Таким образом, большинство рассуждений, использующих термодинамические свойства энтропии применительно к информации нашей реальности, носят спекулятивный характер. В частности, являются необоснованными использование понятия «энтропия» для систем с конечным и небольшим числом состояний, а также попытки расширительного методологического толкования результатов теории вне довольно примитивных механических моделей, для которых они были получены. Энтропия и неэнтропия - интегральные характеристики протекания стохастических процессов - лишь параллельны информации и превращаются в нее в частном случае.

Информацию следует считать особым видом ресурса, при этом имеется в виду толкование «ресурса» как запаса неких знаний материальных предметов или энергетических, структурных или каких-либо других характеристик предмета. В отличие от ресурсов, связанных с материальными предметами, информационные ресурсы являются неистощимыми и предполагают существенно иные методы воспроизведения и обновления, чем материальные ресурсы.

Рассмотрим некоторый набор свойств информации:

- запоминаемость;
- передаваемость;
- преобразуемость;
- воспроизводимость;
- стираемость.

Свойство **запоминаемости** - одно из самых важных. Запоминаемую информацию будем называть макроскопической (имея в виду пространственные масштабы запоминающей ячейки и время запоминания). Именно с макроскопической информацией мы имеем дело в реальной практике.

Передаваемость информации с помощью каналов связи (в том числе с помехами) хорошо исследована в рамках теории информации К. Шеннона. В данном случае имеется ввиду несколько иной аспект - способность информации к копированию, т.е. к тому, что она может быть «запомнена» другой макроскопической системой и при этом останется тождественной самой себе. Очевидно, что количество информации не должно возрастать при копировании.

Воспроизводимость информации тесно связана с ее передаваемостью и не является ее независимым базовым свойством. Если передаваемость означает, что не следует считать существенными пространственные отношения между частями системы, между которыми передается информация, то воспроизводимость характеризует неиссякаемость и неистощимость информации, т.е. что при копировании информация остается тождественной самой себе.

Фундаментальное свойство информации - **преобразуемость**. Оно означает, что информация может менять способ и форму своего существования. Копируемость есть разновидность преобразования информации, при котором ее количество не меняется. В общем случае количество информации в процессах преобразования меняется, но возрастать не может. Свойство **стираемости** информации также не является независимым. Оно связано с таким преобразованием информации (передачей), при котором ее количество уменьшается и становится равным нулю.

Данных свойств информации недостаточно для формирования ее меры, так как они относятся к физическому уровню информационных процессов.

Подводя итог сказанному в п. 2.4 - 2.5, отметим, что предпринимаются (но отнюдь не завершены) усилия ученых, представляющих самые разные области знания, построить единую теорию, которая призвана формализовать понятие информации и информационного процесса, описать превращения информации в процессах самой разной природы. Движение информации есть сущность процессов управления, которые суть проявление имманентной активности материи, ее способности к самодвижению. С момента возникновения кибернетики управление рассматривается применительно ко всем формам движения материи, а не только к высшим (биологической и социальной). Многие проявления движения в неживых - искусственных (технических) и естественных (природных) - системах также обладают общими признаками управления, хотя их исследуют в химии, физике, механике в энергетической, а не в информационной системе представлений. Информационные аспекты в таких системах составляют предмет новой междисциплинарной науки - синергетики.

Высшей формой информации, проявляющейся в управлении в социальных системах, являются знания. Это наддисциплинарное понятие, широко используемое в педагогике и исследованиях по искусственному интеллекту, также претендует на роль важнейшей философской категории. В философском плане познание следует рассматривать как один из функциональных аспектов управления. Такой подход открывает путь к системному пониманию генезиса процессов познания, его основ и перспектив.

Контрольные вопросы

1. Какая форма представления информации - непрерывная или дискретная - приемлема для компьютеров и почему?
2. В чем состоит процедура дискретизации непрерывной информации?
3. Как определяется понятие энтропии?
4. Каким образом определяется единица количества информации при кибернетическом подходе?
5. Каковы особенности определения количества информации, связанной с появлением различных знаков в сообщениях?

§ 3. СИСТЕМЫ СЧИСЛЕНИЯ

3.1. ПОЗИЦИОННЫЕ СИСТЕМЫ СЧИСЛЕНИЯ

Система счисления - принятый способ записи чисел и сопоставления этим записям реальных значений. Все системы счисления можно разделить на два класса: **позиционные и непозиционные**. Для записи чисел в различных системах счисления используется некоторое количество от-

личных друг от друга знаков. Число таких знаков в позиционной системе счисления называется **основанием системы счисления**. Ниже приведена табл. 1.4, содержащая наименования некоторых позиционных систем счисления и перечень знаков (цифр), из которых образуются в них числа.

Таблица 1.4. Некоторые системы счисления

Основание	Система счисления	Знаки
2	Двоичная	0,1
3	Троичная	0,1,2
4	Четвертичная	0,1,2,3
5	Пятиричная	0,1,2,3,4
8	Восьмиричная	0,1,2,3,4,5,6,7
10	Десятичная	0,1,2,3,4,5,6,7,8,9
12	Двенадцатиричная	0,1,2,3,4,5,6,7,8,9,A,B
16	Шестнадцатиричная	0,1,2,3,4,5,6,7,8,9,A,B,D,E,F

В позиционной системе счисления число может быть представлено в виде суммы произведений коэффициентов на степени основания системы счисления:

$$A_n A_{n-1} A_{n-2} \dots A_1 A_0 A_{-1} A_{-2} = A_n B^n + A_{n-1} B^{n-1} + \dots + A_1 B^1 + A_0 B^0 + A_{-1} B^{-1} + A_{-2} B^{-2} + \dots$$

(знак «точка» отделяет целую часть числа от дробной; знак «звездочка» здесь и ниже используется для обозначения операции умножения). Таким образом, значение каждого знака в числе зависит от позиции, которую занимает знак в записи числа. Именно поэтому такие системы счисления называют позиционными. Примеры (десятичный индекс внизу указывает основание системы счисления):

$$23,43_{(10)} = 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 3 \cdot 10^{-2}$$

(в данном примере знак «3» в одном случае означает число единиц, а в другом - число сотых долей единицы);

$$692_{(10)} = 6 \cdot 10^2 + 9 \cdot 10^1 + 2.$$

(«Шестьсот девяносто два» с формальной точки зрения представляется в виде «шесть умножить на десять в степени два, плюс девять умножить на десять в степени один, плюс два»).

$$\begin{aligned} 1101_{(2)} &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0; \\ 112_{(3)} &= 1 \cdot 3^2 + 1 \cdot 3^1 + 2 \cdot 3^0; \\ 341,5_{(8)} &= 3 \cdot 8^2 + 4 \cdot 8^1 + 1 \cdot 8^0 + 5 \cdot 8^{-1}; \\ A1F4_{(16)} &= A \cdot 16^3 + 1 \cdot 16^2 + F \cdot 16^1 + 4 \cdot 16^0. \end{aligned}$$

При работе с компьютерами приходится параллельно использовать несколько позиционных систем счисления (чаще всего двоичную, десятичную и шестнадцатиричную), поэтому большое практическое значение имеют процедуры перевода чисел из одной системы счисления в другую. Заметим, что во всех приведенных выше примерах результат является десятичным числом, и, таким образом, способ перевода чисел из любой позиционной системы счисления в десятичную уже продемонстрирован.

Чтобы перевести целую часть числа из десятичной системы в систему с основанием B ,

необходимо разделить ее на B . Остаток даст младший разряд числа. Полученное при этом частное необходимо вновь разделить на B - остаток даст следующий разряд числа и т.д. Для перевода дробной части ее необходимо умножить на B . Целая часть полученного произведения будет первым (после запятой, отделяющей целую часть от дробной) знаком. Дробную же часть произведения необходимо вновь умножить на B . Целая часть полученного числа будет следующим знаком и т.д.

Отметим, что кроме рассмотренных выше позиционных систем счисления существуют такие, в которых значение знака не зависит от того места, которое он занимает в числе. Такие системы счисления называются непозиционными. Наиболее известным примером непозиционной системы является римская. В этой системе используется 7 знаков (I, V, X, L, C, D, M), которые соответствуют следующим величинам:

1(1)	V(5)	X(10)	L(50)	C (100)	D(500)	M(1000)

Примеры: III (три), LIX (пятьдесят девять), DLV (пятьсот пятьдесят пять).

Недостатком непозиционных систем, из-за которых они представляют лишь исторический интерес, является отсутствие формальных правил записи чисел и, соответственно, арифметических действий над ними (хотя по традиции римскими числами часто пользуются при нумерации глав в книгах, веков в истории и др.).

3.2. ДВОИЧНАЯ СИСТЕМА СЧИСЛЕНИЯ

Особая значимость двоичной системы счисления в информатике определяется тем, что внутреннее представление любой информации в компьютере является двоичным, т.е. описываемым наборами только из двух знаков (0 и 1).

Конкретизируем описанный выше способ в случае перевода чисел из десятичной системы в двоичную. Целая и дробная части переводятся порознь. Для перевода целой части (или просто целого) числа необходимо разделить ее на основание системы счисления и продолжать делить частные от деления до тех пор, пока частное не станет равным 0. Значения получившихся остатков, взятые в обратной последовательности, образуют искомое двоичное число. Например:

	Остаток	
25 : 2 = 12		(1),
12 : 2 = 6		(0),
6 : 2 = 3		(0),
3 : 2 = 1		(1),
1 : 2 = 0		(1).

Таким образом

$$25_{(10)} = 11001_{(2)}.$$

Для перевода дробной части (или числа, у которого «0» целых) надо умножить ее на 2. Целая часть произведения будет первой цифрой числа в двоичной системе. Затем, отбрасывая у результата целую часть, вновь умножаем на 2 и т.д. Заметим, что конечная десятичная дробь при этом вполне может стать бесконечной {периодической} двоичной. Например:

$0,73 \cdot 2 = 1,46$ (целая часть 1),
 $0,46 \cdot 2 = 0,92$ (целая часть 0),
 $0,92 \cdot 2 = 1,84$ (целая часть 1),
 $0,84 \cdot 2 = 1,68$ (целая часть 1) и т.д.

В итоге

$$0,73_{(10)} = 0,1011..._{(2)}.$$

Над числами, записанными в любой системе счисления, можно; производить различные арифметические операции. Так, для сложения и умножения двоичных чисел необходимо использовать табл. 1.5.

Таблица 1.5. Таблицы сложения и умножения в двоичной системе

+	0	1
0	0	1
1	1	10

*	0	1
0	0	0
1	0	1

Заметим, что при двоичном сложении $1 + 1$ возникает перенос единицы в старший разряд - точь-в-точь как в десятичной арифметике:

$$\begin{array}{r} 1001 \\ + \quad 11 \\ \hline = 1100 \end{array}$$

$$\begin{array}{r} 1001 \\ * \quad 11 \\ \hline 111 \\ + \quad 111 \\ \hline = 10101 \end{array}$$

3.3. ВОСЬМЕРИЧНАЯ И ШЕСТНАДЦАТИРИЧНАЯ СИСТЕМЫ СЧИСЛЕНИЯ

С точки зрения изучения принципов представления и обработки информации в компьютере, обсуждаемые в этом пункте системы представляют большой интерес.

Хотя компьютер «знает» только двоичную систему счисления, часто с целью уменьшения количества записываемых на бумаге или вводимых с клавиатуры компьютера знаков бывает удобнее пользоваться восьмеричными или шестнадцатиричными числами, тем более что, как будет показано далее, процедура взаимного перевода чисел из каждой из этих систем в двоичную очень проста - гораздо проще переводов между любой из этих трех систем и десятичной.

Перевод чисел из десятичной системы счисления в восьмеричную производится (по аналогии с двоичной системой счисления) с помощью делений и умножений на 8. Например, переведем число $58,32_{(10)}$:

$$\begin{aligned} 58 : 8 &= 7 \quad (2 \text{ в остатке}), \\ 7 : 8 &= 0 \quad (7 \text{ в остатке}). \\ 0,32 \cdot 8 &= 2,56, \\ 0,56 \cdot 8 &= 4,48, \\ 0,48 \cdot 8 &= 3,84, \dots \end{aligned}$$

Таким образом,

$$58,32_{(10)} = 72,243..._{(8)}$$

(из конечной дроби в одной системе может получиться бесконечная дробь в другой).

Перевод чисел из десятичной системы счисления в шестнадцатиричную производится аналогично.

С практической точки зрения представляет интерес процедура взаимного преобразования двоичных, восьмеричных и шестнадцатиричных чисел. Для этого воспользуемся табл. 1.6 чисел от 0 до 15 (в десятичной системе счисления), представленных в других системах счисления.

Для перевода целого двоичного числа в восьмеричное необходимо разбить его справа налево на группы по 3 цифры (самая левая группа может содержать менее трех двоичных цифр), а затем каждой группе поставить в соответствие ее восьмеричный эквивалент. Например:

$11011001 = 11011001$, т.е. $11011001_{(2)} = 331_{(8)}$.

Заметим, что группу из трех двоичных цифр часто называют «двоичной триадой».

Перевод целого двоичного числа в шестнадцатеричное производится путем разбиения данного числа на группы по 4 цифры - «двоичные тетрады»:

$1100011011001 = 1\ 1000\ 1101\ 1001$, т.е. $1100011011001_{(2)} = 18D9_{(16)}$.

Для перевода дробных частей двоичных чисел в восьмеричную или шестнадцатеричную системы аналогичное разбиение на триады или тетрады производится от точки вправо (с дополнением недостающих последних цифр нулями):

$0,1100011101_{(2)} = 0,110\ 001\ 110\ 100 = 0,6164_{(8)}$,

$0,1100011101_{(2)} = 0,1100\ 0111\ 0100 = 0,C74_{(16)}$.

Перевод восьмеричных (шестнадцатеричных) чисел в двоичные производится обратным путем - сопоставлением каждому знаку числа соответствующей тройки (четверки) двоичных цифр.

Таблица 1.6 Соответствие чисел в различных системах счисления

Десятичная	Шестнадцатеричная	Восьмеричная	Двоичная
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Преобразования чисел из двоичной в восьмеричную и шестнадцатеричную системы и наоборот столь просты (по сравнению с операциями между этими тремя системами и привычной нам десятичной) потому, что числа 8 и 16 являются целыми степенями числа 2. Этой простотой и объясняется популярность восьмеричной и шестнадцатеричной систем в вычислительной технике и программировании.

Арифметические действия с числами в восьмеричной и шестнадцатеричной системах счисления выполняются по аналогии с двоичной и десятичной системами. Для этого необходимо воспользоваться соответствующими таблицами. Для примера табл. 1.7 иллюстрирует сложение и умножение восьмеричных чисел.

Рассмотрим еще один возможный способ перевода чисел из одной позиционной системы счисления в другую - *метод вычитания степеней*. В этом случае из числа последовательно вычитается максимально допустимая степень требуемого основания, умноженная на максимально возможный коэффициент, меньший основания; этот коэффициент и является значащей цифрой числа в новой системе. Например, число $114_{(10)}$:

$$114 - 2^6 = 114 - 64 = 50,$$

$$50 - 2^5 = 50 - 32 = 18,$$

$$18 - 2^4 = 2,$$

$$2 - 2^1 = 0.$$

Таким образом, $114_{(10)} = 1110010_{(2)}$.

$$114 - 1 \cdot 8^2 = 114 - 64 = 50,$$

$$50 - 6 \cdot 8^1 = 50 - 48 = 2,$$

$$2 - 2 \cdot 8^0 = 2 - 2 = 0.$$

Итак, $114_{(10)} = 162_{(8)}$.

Таблица 1.7 Таблицы сложения и умножения в восьмеричной системе

Сложение

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Умножение

*	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Контрольные вопросы

1. В чем отличие позиционной системы счисления от непозиционной?
2. Каковы способы перевода чисел из одной системы счисления в другую?
3. В чем заключается преимущество использования восьмеричной и шестнадцатеричной систем счисления в вычислительной технике?
4. Как выглядят таблицы сложения и умножения в шестнадцатеричной системе?

§ 4. КОДИРОВАНИЕ ИНФОРМАЦИИ.

4.1. АБСТРАКТНЫЙ АЛФАВИТ

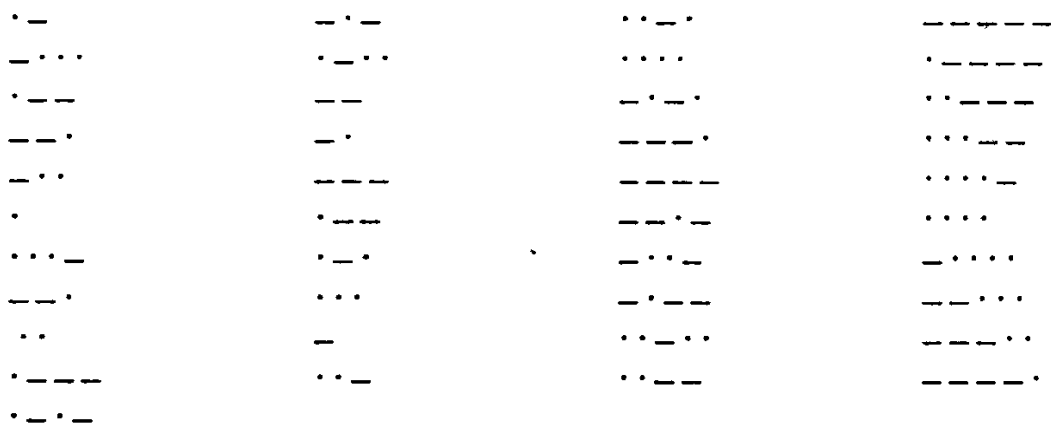
Информация передается в виде сообщений. Дискретная информация записывается с помощью некоторого конечного набора знаков, которые будем называть буквами, не вкладывая в это слово привычного ограниченного значения (типа «русские буквы» или «латинские буквы»). Буква в данном расширенном понимании - любой из знаков, которые некоторым соглашением установлены для общения. Например, при привычной передаче сообщений на русском языке такими знаками будут русские буквы - прописные и строчные, знаки препинания, пробел; если в тексте есть числа - то и цифры. Вообще, буквой будем называть элемент некоторого конечного множества (набора) отличных друг от друга знаков. Множество знаков, в котором определен их порядок, назовем алфавитом (общеизвестен порядок знаков в русском алфавите: А, Б,..., Я).

Рассмотрим некоторые примеры алфавитов.

1. Алфавит прописных русских букв:

А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Ь Э Ю Я

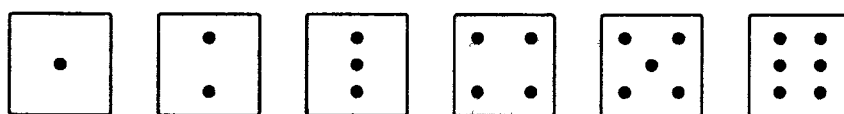
2. Алфавит Морзе:



3. Алфавит клавиатурных символов ПЭВМ IBM (русифицированная клавиатура):

```
` 1 2 3 4 5 6 7 8 9 0 - =
~ ! @ # $ % ^ & * ( ) _ +
q w e r t y u i o p [ ]
Q W E R T Y U I O P { }
a s d f g h j k l ; : "
A S D F G H J K L : « »
z x c v b n m , . /
Z X C V B N M < > ?
й ц у к е н г ш щ з х ъ
Й Ц У К Е Н Г Ш Щ З Х Ъ
ф ы в а п р о л д ж э
Ф Ы В А П Р О Л Д Ж Э
я ч с м и т ь б ю
Я Ч С М И Т Ь Б Ю
```

4. Алфавит знаков правильной шестигранной игральной кости:



5. Алфавит арабских цифр:

0 1 2 3 4 5 6 7 8 9

6. Алфавит шестнадцатичных цифр:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Этот пример, в частности, показывает, что знаки одного алфавита могут образовываться из знаков других алфавитов.

7. Алфавит двоичных цифр:

0 1

Алфавит 7 является одним из примеров, так называемых, «двоичных» алфавитов, т.е. алфавитов, состоящих из двух знаков. Другими примерами являются двоичные алфавиты 8 и 9:

8. Двоичный алфавит «точка, «тире»: . _

9. Двоичный алфавит «плюс», «минус»: + -

10. Алфавит прописных латинских букв:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

11. Алфавит римской системы счисления:

I V X L C D M

12. Алфавит языка блок-схем изображения алгоритмов:



13. Алфавит языка программирования Паскаль (см. в главе 3).

4.2. КОДИРОВАНИЕ И ДЕКОДИРОВАНИЕ

В канале связи сообщение, составленное из символов (букв) одного алфавита, может пре-

образовываться в сообщение из символов (букв) другого алфавита. Правило, описывающее однозначное соответствие букв алфавитов при таком преобразовании, называют кодом. Саму процедуру преобразования сообщения называют перекодировкой. Подобное преобразование сообщения может осуществляться в момент поступления сообщения от источника в канал связи (кодирование) и в момент приема сообщения получателем (декодирование). Устройства, обеспечивающие кодирование и декодирование, будем называть соответственно кодировщиком и декодировщиком. На рис. 1.5 приведена схема, иллюстрирующая процесс передачи сообщения в случае перекодировки, а также воздействия помех (см. следующий пункт).



Рис. 1.5. Процесс передачи сообщения от источника к приемнику

Рассмотрим некоторые примеры кодов.

1. Азбука Морзе в русском варианте (алфавиту, составленному из алфавита русских заглавных букв и алфавита арабских цифр ставится в соответствие алфавит Морзе):

А	..-	К	---.	Ф		0	-----
Б		Л		Х		1	.-----
В	..--	М	--	Ц	..--	2	..----
Г	---	Н	---	Ч	---	3	...---
Д		О	----	Ш	----	4	-
Е	..	П		Щ		5	
Ж	...-	Р		Ъ, Ъ	...-	6	
З	---	С	...	Ы	...-	7	
И	..	Т	-	Э		8	
Й		У	...-	Ю	...-	9	
		Я	...-				

2. Код Трисиме (знакам латинского алфавита ставятся в соответствие комбинации из трех знаков: 1,2,3):

А 111	Д 121	Г 131	Ж 211	М 221	Р 231	С 311	В 321	У 331
В 112	Е 122	Н 132	К 212	О 222	П 232	Т 312	Х 322	Ц 332
С 113	Ф 123	И 133	Л 213	Ш 223	Щ 233	У 313	Х 323	Ц 333

Код Трисиме является примером, так называемого, равномерного кода (такого, в котором все кодовые комбинации содержат одинаковое число знаков - в данном случае три). Пример неравномерного кода - азбука Морзе.

3. Кодирование чисел знаками различных систем счисления см. §3.

4.3. ПОНЯТИЕ О ТЕОРЕМАХ ШЕННОНА

Ранее отмечалось, что при передаче сообщений по каналам связи могут возникать помехи, способные привести к искажению принимаемых знаков. Так, например, если вы попытаетесь в ветреную погоду передать речевое сообщение человеку, находящемуся от вас на значительном расстоянии, то оно может быть сильно искажено такой помехой, как ветер. Вообще, передача сообщений при наличии помех является серьезной теоретической и практической задачей. Ее значимость возрастает в связи с повсеместным внедрением компьютерных телекоммуникаций, в которых помехи неизбежны. При работе с кодированной информацией, искажаемой помехами, можно выделить следующие основные проблемы: установления самого факта того, что произошло искажение информации; выяснения того, в каком конкретно месте передаваемого текста это произошло; исправления ошибки, хотя бы с некоторой степенью достоверности.

Помехи в передаче информации - вполне обычное дело во всех сферах профессиональной деятельности и в быту. Один из примеров был приведен выше, другие примеры - разговор по телефону, в трубке которого «трещит», вождение автомобиля в тумане и т.д. Чаще всего человек вполне справляется с каждой из указанных выше задач, хотя и не всегда отдает себе отчет, как он это делает (т.е. неалгоритмически, а исходя из каких-то ассоциативных связей). Известно, что

естественный язык обладает большой **избыточностью** (в европейских языках - до 7%), чем объясняется большая помехоустойчивость сообщений, составленных из знаков алфавитов таких языков. Примером, иллюстрирующим устойчивость русского языка к помехам, может служить предложение «в словах все гласное заменено буквой о». Здесь 26% символов «поражены», однако это не приводит к потере смысла. Таким образом, в данном случае избыточность является полезным свойством.

Избыточность могла бы быть использована и при передаче кодированных сообщений в технических системах. Например, каждый фрагмент текста («предложение») передается трижды, и верным считается та пара фрагментов, которая полностью совпала. Однако, большая избыточность приводит к большим временным затратам при передаче информации и требует большого объема памяти при ее хранении. Впервые теоретическое исследование эффективного кодирования предпринял К.Шеннон.

Первая теорема Шеннона декларирует возможность создания системы эффективного кодирования дискретных сообщений, у которой среднее число двоичных символов на один символ сообщения асимптотически стремится к энтропии источника сообщений (в отсутствии помех).

Задача эффективного кодирования описывается триадой:

$$X = \{x_i\} - \text{кодирующее устройство} - B.$$

Здесь X, B - соответственно входной и выходной алфавит. Под множеством x_i можно понимать любые знаки (буквы, слова, предложения). B - множество, число элементов которого в случае кодирования знаков числами определяется основанием системы счисления (например, $m = 2$). Кодировщик сопоставляет каждому сообщению x_i из X кодовую комбинацию, составленную из n_i символов множества B . Ограничением данной задачи является отсутствие помех. Требуется оценить минимальную среднюю длину кодовой комбинации.

Для решения данной задачи должна быть известна вероятность P_i появления сообщения x_i , которому соответствует определенное количество символов n_i алфавита B . Тогда математическое ожидание количества символов из B определится следующим образом:

$$n_{cp} = \sum n_i P_i \text{ (средняя величина).}$$

Этому среднему числу символов алфавита B соответствует максимальная энтропия $H_{max} = n_{cp} \log m$. Для обеспечения передачи информации, содержащейся в сообщениях X кодовыми комбинациями из B , должно выполняться условие $H_{max} \geq H(x)$, или $n_{cp} \log m \geq - \sum P_i \log P_i$. В этом случае закодированное сообщение имеет избыточность $n_{cp} \geq H(x) / \log m$, $n_{min} = H(x) / \log m$.

Коэффициент избыточности

$$K_u = (H_{max} - H(x)) / H_{max} = (n_{cp} - n_{min}) / n_{cp}$$

Выпишем эти значения в виде табл. 1.8. Имеем:

$$n_{min} = H(x) / \log 2 = 2,85, \quad K_u = (2,92 - 2,85) / 2,92 = 0,024,$$

т.е. код практически не имеет избыточности. Видно, что среднее число двоичных символов стремится к энтропии источника сообщений.

Таблица 1.8 Пример к первой теореме Шеннона

N	Px_i	x_i	Код	n_i	$n_i \cdot P_i$	$Px_i \cdot \log Px_i$
1	0,19	X_1	10	2	0,38	-4,5522
2	0,16	X_2	001	3	0,48	-4,2301
3	0,16	X_3	011	3	0,48	-4,2301
4	0,15	X_4	101	3	0,45	-4,1054
5	0,12	X_5	111	3	0,36	-3,6706

6	0,11	X_6	111	3	0,33	- 3,5028
7	0,09	X_7	1011	4	0,36	-3,1265
8	0,02	X_8	1001	4	0,08	-3,1288
$\Sigma=1$					$\Sigma=2,92$	$\Sigma=2,85$

Вторая теорема Шеннона гласит, что при наличии помех в канале всегда можно найти такую систему кодирования, при которой сообщения будут переданы с заданной достоверностью. При наличии ограничения пропускная способность канала должна превышать производительность источника сообщений. Таким образом, вторая теорема Шеннона устанавливает принципы помехоустойчивого кодирования. Для дискретного канала с помехами теорема утверждает, что, если скорость создания сообщений меньше или равна пропускной способности канала, то существует код, обеспечивающий передачу со сколь угодно малой частотой ошибок.

Доказательство теоремы основывается на следующих рассуждениях. Первоначально последовательность $X = \{x_i\}$ кодируется символами из B так, что достигается максимальная пропускная способность (канал не имеет помех). Затем в последовательность из B длины n вводится r символов и по каналу передается новая последовательность из $n + r$ символов. Число возможных последовательностей длины $n + r$ больше числа возможных последовательностей длины n . Множество всех последовательностей длины $n + r$ может быть разбито на n подмножеств, каждому из которых сопоставлена одна из последовательностей длины n . При наличии помехи на последовательность из $n + r$ выводит ее из соответствующего подмножества с вероятностью сколь угодно малой.

Это позволяет определять на приемной стороне канала, какому подмножеству принадлежит искаженная помехами принятая последовательность длины $n + r$, и тем самым восстановить исходную последовательность длины n .

Эта теорема не дает конкретного метода построения кода, но указывает на пределы достижимого в создании помехоустойчивых кодов, стимулирует поиск новых путей решения этой проблемы.

4.4. МЕЖДУНАРОДНЫЕ СИСТЕМЫ БАЙТОВОГО КОДИРОВАНИЯ

Информатика и ее приложения интернациональны. Это связано как с объективными потребностями человечества в единых правилах и законах хранения, передачи и обработки информации, так и с тем, что в этой сфере деятельности (особенно в ее прикладной части) заметен приоритет одной страны, которая благодаря этому получает возможность «диктовать моду».

Компьютер считают универсальным преобразователем информации. Тексты на естественных языках и числа, математические и специальные символы - одним словом все, что в быту или в профессиональной деятельности может быть необходимо человеку, должно иметь возможность быть введенным в компьютер.

В силу безусловного приоритета двоичной системы счисления при внутреннем представлении информации в компьютере кодирование «внешних» символов основывается на сопоставлении каждому из них определенной группы двоичных знаков. При этом из технических соображений и из соображений удобства кодирования-декодирования следует пользоваться равномерными кодами, т.е. двоичными группами равной длины.

Попробуем подсчитать наиболее короткую длину такой комбинации с точки зрения человека, заинтересованного в использовании лишь одного естественного алфавита - скажем, английского: 26 букв следует умножить на 2 (прописные и строчные) - итого 52; 10 цифр, будем считать, 10 знаков препинания; 10 разделительных знаков (три вида скобок, пробел и др.), знаки привычных математических действий, несколько специальных символов (типа #, \$, & и др.) — итого ~ 100. Точный подсчет здесь не нужен, поскольку нам предстоит решить простейшую задачу: имея, скажем, равномерный код из групп по N двоичных знаков, сколько можно образовать разных кодовых комбинаций. Ответ очевиден $K = 2^N$. Итак, при $N = 6$ $K = 64$ - явно мало, при $N = 7$ $K = 128$ - вполне достаточно.

Однако, для кодирования нескольких (хотя бы двух) естественных алфавитов (плюс все отмеченные выше знаки) и этого недостаточно. Минимально достаточное значение N в этом случае 8; имея 256 комбинаций двоичных символов, вполне можно решить указанную задачу. Поскольку 8 двоичных символов составляют 1 байт, то говорят о системах «байтового» кодирования.

Наиболее распространены две такие системы: EBCDIC (Extended Binary Coded Decimal Interchange Code) и ASCII (American Standard Information Interchange).

Первая - исторически тяготеет к «большим» машинам, вторая чаще используется на мини- и микро-ЭВМ (включая персональные компьютеры). Ознакомимся подробнее именно с ASCII, созданной в 1963 г.

В своей первоначальной версии это - система семибитного кодирования. Она ограничивалась одним естественным алфавитом (английским), цифрами и набором различных символов, включая «символы пишущей машинки» (привычные знаки препинания, знаки математических действий и др.) и «управляющие символы». Примеры последних легко найти на клавиатуре компьютера: для микро-ЭВМ, например, DEL - знак удаления символа.

В следующей версии фирма IBM перешла на расширенную 8-битную кодировку. В ней первые 128 символов совпадают с исходными и имеют коды со старшим битом равным нулю, а остальные коды отданы под буквы некоторых европейских языков, в основе которых лежит латиница, греческие буквы, математические символы (скажем, знак квадратного корня) и символы псевдографики. С помощью последних можно создавать таблицы, несложные схемы и др.

Для представления букв русского языка (кириллицы) в рамках ASCII было предложено несколько версий. Первоначально был разработан ГОСТ под названием КОИ-7, оказавшийся по ряду причин крайне неудачным; ныне он практически не используется.

В табл. 1.9 приведена часто используемая в нашей стране модифицированная альтернативная кодировка. В левую часть входят исходные коды ASCII; в правую часть (расширение ASCII) вставлены буквы кириллицы взамен букв, немецкого, французского алфавитов (не совпадающих по написанию с английскими), греческих букв, некоторых спецсимволов.

Знакам алфавита ПЭВМ ставятся в соответствии шестнадцатиричные числа по правилу: первая - номер столбца, вторая - номер строки. Например: английская 'A' - код 41, русская 'и' - код A8.

Таблица 1.9 Таблица кодов ASCII (расширенная)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4																
5																
6																
7																
8																
9																
A																
B																
C																
D																
E																
F																

Одним из достоинств этой системы кодировки русских букв является их естественное упорядочение, т.е. номера букв следуют друг за другом в том же порядке, в каком сами буквы стоят в русском алфавите. Это очень существенно при решении ряда задач обработки текстов, когда требуется выполнить или использовать лексикографическое упорядочение слов.

Из сказанного выше следует, что даже 8-битная кодировка недостаточна для кодирования всех символов, которые хотелось бы иметь в расширенном алфавите. Все препятствия могут быть сняты при переходе на 16-битную кодировку Unicode, допускающую 65536 кодовых комбинаций.

Контрольные вопросы

1. Как определяется алфавит?
2. Что такое код?
3. Как объяснить большую помехоустойчивость передаваемых сообщений, составленных на русском языке?
4. Что определяют первая и вторая теоремы Шеннона?

§ 5. ЭЛЕМЕНТЫ ТЕОРИИ ГРАФОВ

5.1. ОСНОВНЫЕ ПОНЯТИЯ

Такая структура, как **граф** (в качестве синонима используется также термин «сеть»), имеет самые различные применения в информатике и в смежных прикладных областях, поэтому познакомимся с основными понятиями теории графов.

Граф $G = (V, E)$ задается парой конечных множеств V и E . Элементы первого множества $v_1, v_2, \dots, v_M$ называются **вершинами** графа (при графическом представлении им соответствуют точки). Элементы второго множества $e_1, e_2, \dots, e_N$ называют **ребрами**. Каждое ребро определяется парой вершин (при графическом представлении ребро соединяет две вершины графа). Если ребра графа определяются упорядоченными парами вершин, то такой граф называют **ориентированным** (на чертеже при изображении ориентированного графа на каждом ребре ставят стрелку, указывающую его направление). Ориентированный граф с пятью вершинами и семью ребрами изображен на рис. 1.6.

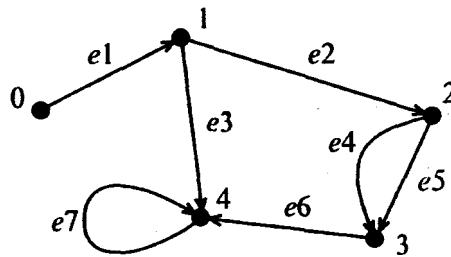


Рис. 1.6. Пример ориентированного графа

Если две вершины соединены двумя или более ребрами, то эти ребра называют параллельными (например, ребра e_4 и e_5). Если начало и конец ребра совпадают, то такое ребро называется **петлей** (например, ребро e_7). Граф без петель и параллельных ребер называется простым.

Если ребро e_k определяется вершинами v_i и v_j (будем обозначать этот факт следующим образом: $e_k = (v_i, v_j)$), то говорят, что ребро e_k **инцидентно** вершинам v_i и v_j . Две вершины v_i и v_j называются смежными, если в графе существует ребро (v_i, v_j) .

Последовательность вершин $v_{i1}, v_{i2}, \dots, v_{ik}$, таких, что каждая пара $(v_{i,j-1}, v_{ij})$ при $1 < j \leq k$ определяет ребро, называется **маршрутом** в графе G . Вершины v_{i1} и v_{ik} называют **концевыми** вершинами маршрута, все остальные входящие в него вершины - **внутренними**.

Маршрут, в котором все определяемые им ребра различны, называют цепью. Цепь считают замкнутой, если ее концевые вершины совпадают. Замкнутая цепь, в которой все вершины (за исключением концевых) различны, называется циклом. Незамкнутая цепь, в которой все вершины различны, носит название путь. Если в ориентированном графе существует путь из v_i в v_j , то говорят, что вершина v_j достижима из вершины v_i .

Две вершины v_i и v_j называют связанными в графе G , если в нем существует путь, для которого эти вершины являются концевыми. Граф G называется связным, если каждые две вершины в нем являются связанными. На рис. 1.7 изображен простой неориентированный связный граф.

Последовательность вершин v_1, v_5, i_4, v_3 , например, определяет путь, а последовательность вершин $v_1, v_5, i_4, v_3, v_1, v_1$ - цикл. **Деревом** будем называть неориентированный связный граф без циклов. **Лес** - это любой граф без циклов. На рис. 1.8 показаны возможные деревья с пятью вершинами.

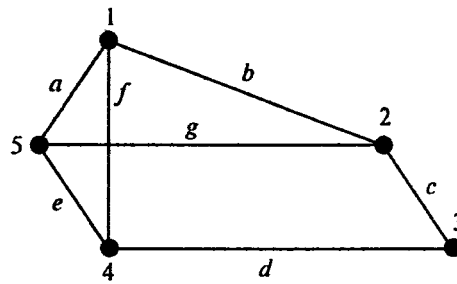


Рис. 1.7. Пример неориентированного связного графа

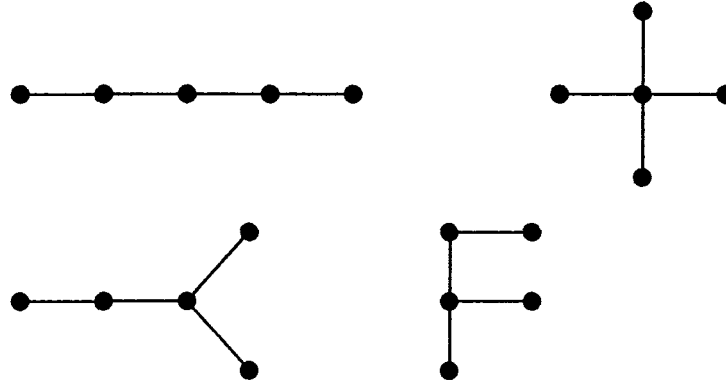


Рис. 1.8. Примеры деревьев

Анализ приведенных здесь понятий и определений показывает, что в качестве моделей графы удобно использовать в тех случаях, когда рассматривается система каких-либо объектов, между которыми существуют определенные связи, отношения, когда изучается структура системы, возможности ее функционирования. В информатике графы используются в разделах: операционные системы, алгоритмизация, структуры данных, информационное моделирование и др.

5.2. ПРЕДСТАВЛЕНИЕ ГРАФОВ

Важным вопросом, особенно для приложений теории графов, является определение возможных способов представления графов. Самый простой способ - полное перечисление множеств V и E . Однако, очевидно, что в этом случае выявление у графа различных характеристик и свойств будет крайне затруднительным. Граф можно представить в виде некоторого графического изображения и визуально определить некоторые свойства и характеристики заданного графа. Однако, при наличии в графе большого числа ребер и вершин этот способ также мало пригоден. Рассматривая различные возможные способы представления графов, мы должны иметь в виду потребность ввода соответствующей информации в компьютер. В этой связи ввод информации в числовом виде предпочтителен, хотя современные технические средства допускают ввод и графической информации (таблиц, текста, графиков, рисунков и т.д.), после чего может производиться обработка такой информации.

Матрица смежности. Если вершины графа G помечены метками $v_1, v_2, \dots, v_n$, то элементы матрицы смежности $A(G)$ размера $V, \times V$ определяются следующим образом: $A(i, j) = 1$, если v_i смежна с v_j ; $A(i, j) = 0$ в противном случае (рис. 1.9, а).

Матрица инцидентности. Если вершины графа G помечены метками $v_1, v_2, \dots, v_m$, а ребра - метками $e_1, e_2, \dots, e_n$, то элементы матрицы инцидентности $I(G)$ размера $M \times N$ определяются правилом: $B(i, j) = 1$, если v_i инцидентна e_j ; $B(i, j) = 0$ в противном случае (см. рис. 1.9, б).

	1	2	3	4	5
1	0	1	0	1	1
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	1	1	0	1	0

Рис. 1.9, а. Матрица смежности

	a	b	c	d	e	f	g
1	1	1	0	0	0	1	0
2	0	1	1	0	0	0	1
3	0	0	1	1	0	0	0
4	0	0	0	1	1	1	0
5	1	0	0	0	1	0	1

Рис. 1.9, б. Матрица инцидентности

Для ориентированного графа G , имеющего N вершин можно рассмотреть матрицу **достижимости** $C(G)$ размера $N \times N$, элементы которой определяются так: $C(I, J) = 1$, если вершина v_j достижима из v_i ; $C(I, J) = 0$ в противном случае. Ниже приведен пример ориентированного графа и его матрицы достижимости, рис. 1.10.

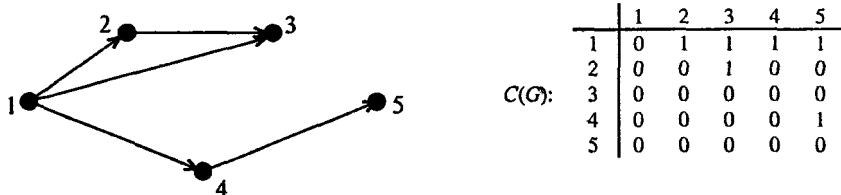


Рис. 1.10. Матрица достижимости ориентированного графа

Контрольные вопросы

1. Каким образом определяется граф?
2. Что является путем в графе?
3. Как определяется такой вид графа, как дерево?
4. Какими способами можно задать граф?

§ 6. АЛГОРИТМ И ЕГО СВОЙСТВА

6.1. РАЗЛИЧНЫЕ ПОДХОДЫ К ПОНЯТИЮ «АЛГОРИТМ»

Понятие алгоритма - одно из фундаментальных понятий информатики. Алгоритмизация наряду с моделированием выступает в качестве общего метода информатики. К реализации определенных алгоритмов сводятся процессы управления в различных системах, что делает понятие алгоритма близким и кибернетике.

Алгоритмы являются объектом систематического исследования пограничной между математикой и информатикой научной дисциплины, примыкающей к математической логике - **теории алгоритмов**.

Особенность положения состоит в том, что при решении практических задач, предполагающих разработку алгоритмов для реализации на ЭВМ, и тем более при использовании на практике информационных технологий, можно, как правило, не опираться на высокую формализацию данного понятия. Поэтому представляется целесообразным познакомиться с алгоритмами и алгоритмизацией на основе содержательного толкования сущности понятия алгоритма и рассмотрения основных его свойств. При таком подходе алгоритмизация более выступает как набор определенных практических приемов, особых специфических навыков рационального мышления в рамках заданных языковых средств. Можно провести аналогию между этим обстоятельством и рассмотренным выше подходом к измерению информации: тонкие математические построения при «кибернетическом» подходе не очень нужны при использовании гораздо более простого «объемного» подхода при практической работе с компьютером.

Само слово «алгоритм» происходит от algorithmi - латинской формы написания имени великого математика IX века аль-Хорезми, который сформулировал правила выполнения арифметических действий. Первоначально под алгоритмами и понимали только правила выполнения четырех арифметических действий над многозначными числами.

6.2. ПОНЯТИЕ ИСПОЛНИТЕЛЯ АЛГОРИТМА

Понятие исполнителя невозможно определить с помощью какой-либо формализации. Исполнителем может быть человек, группа людей, робот, станок, компьютер, язык программирования и т.д. Важнейшим свойством, характеризующим любого из этих исполнителей, является то, что исполнитель умеет выполнять некоторые команды. Так, исполнитель-человек умеет выполнять такие команды как «встать», «сесть», «включить компьютер» и т.д., а исполнитель-язык программирования Бейсик - команды PRINT, END, LIST и другие аналогичные. Вся совокупность ко-

манд, которые данный исполнитель умеет выполнять, называется системой команд **исполнителя** (СКИ).

В качестве примера (рис. 1.11) рассмотрим исполнителя-робота, работа которого состоит в собственном перемещении по рабочему полю (квадрату произвольного размера, разделенному на клетки) и перемещении объектов, в начальный момент времени находящихся на «складе» (правая верхняя клетка).

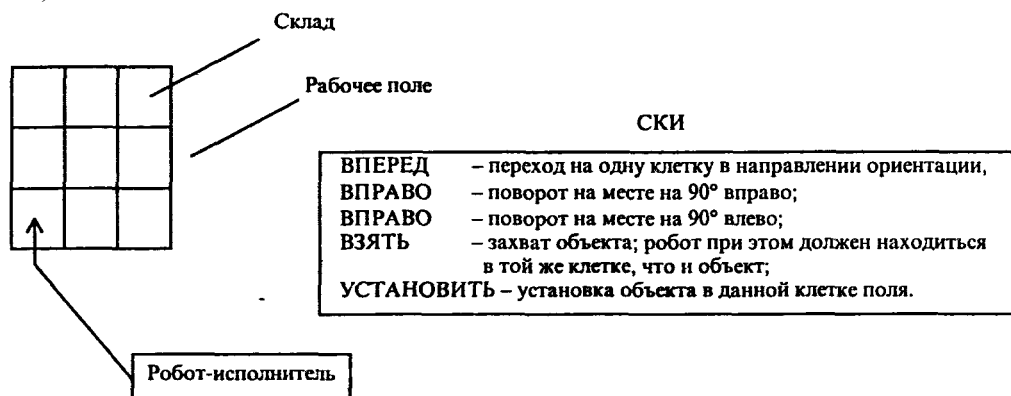


Рис. 1.11. Исполнитель-робот

Одно из принципиальных обстоятельств состоит в том, что исполнитель не вникает в смысл того, что он делает, но получает необходимый результат. В таком случае говорят, что исполнитель действует **формально**, т. е. отвлекается от содержания поставленной задачи и только строго выполняет некоторые правила, инструкции.

Это - важная особенность алгоритмов. Наличие алгоритма формализует процесс решения задачи, исключает рассуждение исполнителя. Использование алгоритма дает возможность решать задачу формально, механически исполняя команды алгоритма в указанной последовательности. Целесообразность предусматриваемых алгоритмом действий обеспечивается точным анализом со стороны того, кто составляет этот алгоритм.

Введение в рассмотрение понятия «исполнитель» позволяет определить алгоритм как понятное и точное предписание исполнителю совершить последовательность действий, направленных на достижение поставленной цели. В случае исполнителя-робота мы имеем пример алгоритма «в обстановке», характеризующегося отсутствием каких-либо величин. Наиболее же распространенными и привычными являются алгоритмы работы с величинами - числовыми, символьными, логическими и т.д.

6.3. ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ АЛГОРИТМОВ

Алгоритм, составленный для некоторого исполнителя, можно представить различными способами: с помощью графического или словесного описания, в виде таблицы, последовательностью формул, записанным на алгоритмическом языке (языке программирования). Остановимся на графическом описании алгоритма, называемом блок-схемой. Этот способ имеет ряд преимуществ благодаря наглядности, обеспечивающей, в частности, высокую «читаемость» алгоритма и явное отображение управления в нем.

Прежде всего определим понятие блок-схемы. Блок-схема - это ориентированный граф, указывающий порядок исполнения команд алгоритма; вершины такого графа могут быть одного из трех типов (рис. 1.12).

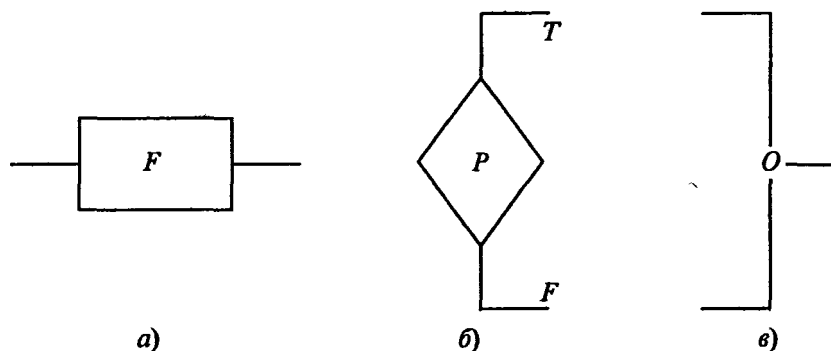


Рис. 1.12. Три типа вершин графа

На рис. 1.12 изображены «функциональная» (а) вершина (имеющая один вход и один выход); «предикатная» (б) вершина, имеющая один вход и два выхода (в этом случае функция P передает управление по одной из ветвей в зависимости от значения P (T , т.е. true, означает «истина», F , т.е. false - «ложь»); «объединяющая» (в) вершина (вершина «слияния»), обеспечивающая передачу управления от одного из двух входов к выходу. Иногда вместо T пишут «да» (либо знак +), вместо F - «нет» (либо знак -).

Из данных элементарных блок-схем можно построить четыре блок-схемы (рис. 1.13), имеющих особое значение для практики алгоритмизации.

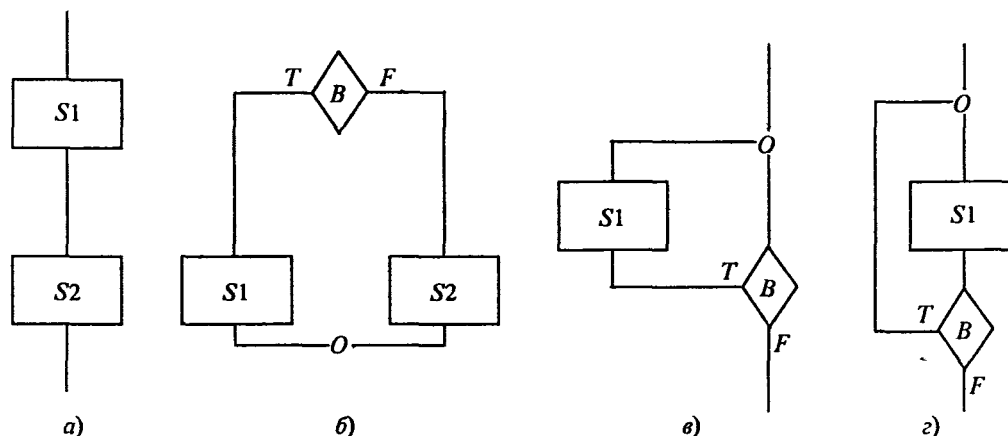


Рис. 1.13. Основные алгоритмические структуры

На рис. 1.13 изображены следующие блок-схемы: а - **композиция**, или следование; б - **альтернатива**, или **развилка**, в и г - блок-схемы, каждую из которых называют **итерацией**, или **циклом** (с предусловием (в), с постусловием (г)). $S1$ и $S2$ представляют собой в общем случае некоторые серии команд для соответствующего исполнителя, B - это условие, в зависимости от истинности (T) или ложности (F) которого управление передаётся по одной из двух ветвей. Можно доказать, что для составления любого алгоритма достаточно представленных выше четырех блок-схем, если пользоваться их последовательностями и/или суперпозициями.

Блок-схема «альтернатива» может иметь и сокращенную форму, в которой отсутствует ветвь $S2$ (рис. 1.14, а). Развитием блок-схемы типа альтернатива является блок-схема «выбор» (рис. 1.14, б).

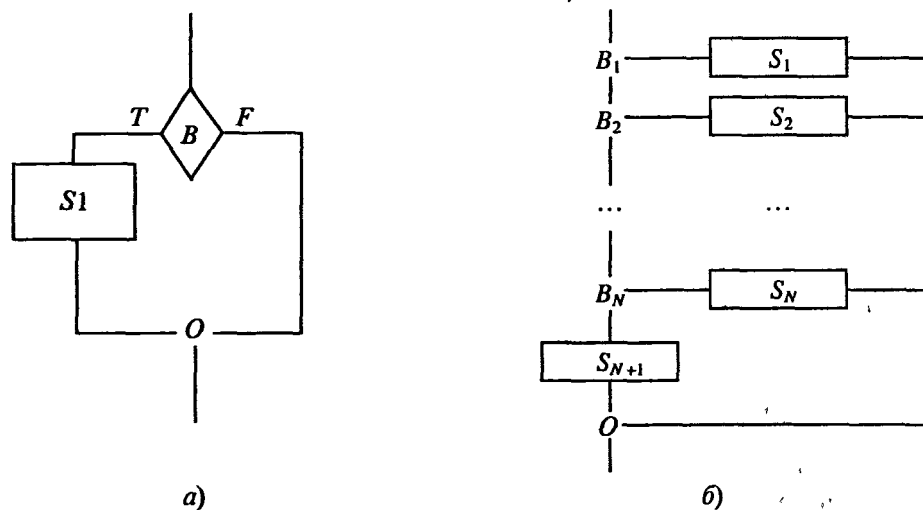


Рис. 1.14. Развитие структуры типа «альтернатива»;
а) - неполная развилка; б) - структура «выбор»

На практике при составлении блок-схем оказывается удобным использовать и другие графические знаки (некоторые из них приведены на рис. 1.15).

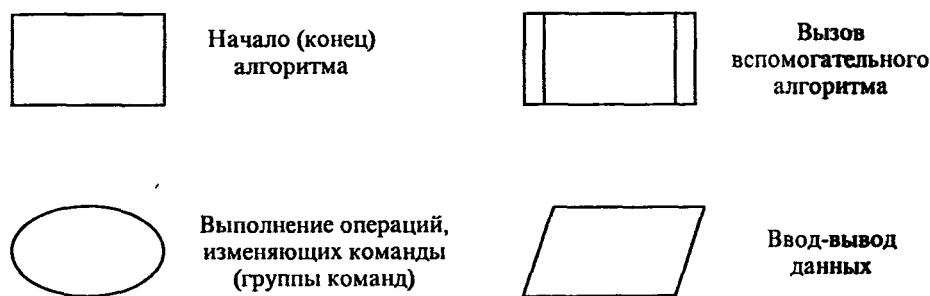


Рис. 1.15. Некоторые дополнительные конструкции для изображения блок-схем алгоритмов

6.4. СВОЙСТВА АЛГОРИТМОВ

Алгоритм должен быть составлен таким образом, чтобы исполнитель, в расчете на которого он создан, мог однозначно и точно следовать командам алгоритма и эффективно получать определенный результат. Это накладывает на записи алгоритмов ряд обязательных требований, суть которых вытекает, вообще говоря, из приведенного выше неформального толкования понятия алгоритма. Сформулируем эти требования в виде перечня свойств, которым должны удовлетворять алгоритмы, адресуемые заданному исполнителю.

1. Одно из первых требований, которое предъявляется к алгоритму, состоит в том, что описываемый процесс должен быть разбит на последовательность отдельных шагов. Возникающая в результате такого разбиения запись представляет собой упорядоченную совокупность четко разделенных друг от друга предписаний (директив, команд, операторов), образующих прерывную (или, как говорят, дискретную) структуру алгоритма. Только выполнив требования одного предписания, можно приступить к выполнению следующего. Дискретная структура алгоритмической записи может, например, подчеркиваться сквозной нумерацией отдельных команд алгоритма, хотя это требование не является обязательным. Рассмотренное свойство алгоритмов называют дискретностью.

2. Используемые на практике алгоритмы составляются с ориентацией на определенного исполнителя. Чтобы составить для него алгоритм, нужно знать, какие команды этот исполнитель может понять и исполнить, а какие - не может. Мы знаем, что у каждого исполнителя имеется своя система команд. Очевидно, составляя запись алгоритма для определенного исполнителя, можно использовать лишь те команды, которые имеются в его СКИ. Это свойство алгоритмов будем

называть понятностью.

3. Будучи понятным, алгоритм не должен содержать предписаний, смысл которых может восприниматься неоднозначно, т.е. одна и та же команда, будучи понятна разным исполнителям, после исполнения каждым из них должна давать одинаковый результат.

Запись алгоритма должна быть настолько четкой, полной и продуманной в деталях, чтобы у исполнителя не могло возникнуть потребности в принятии решений, не предусмотренных составителем алгоритма. Говоря иначе, алгоритм не должен оставлять места для произвола исполнителя. Кроме того, в алгоритмах недопустимы также ситуации, когда после выполнения очередной команды алгоритма исполнителю неясно, какая из команд алгоритма должна выполняться на следующем шаге.

Отмеченные свойства алгоритмов называют **определенностью** или **детерминированностью**.

4. Обязательное требование к алгоритмам - **результативность**. Смысл этого требования состоит в том, что при точном исполнении всех предписаний алгоритма процесс должен прекратиться за конечное число шагов и при этом должен получиться определенный результат. Вывод о том, что решения не существует - тоже результат.

5. Наиболее распространены алгоритмы, обеспечивающие решение не одной конкретной задачи, а некоторого класса задач данного типа. Это свойство алгоритма называют **массовостью**. В простейшем случае массовость обеспечивает возможность использования различных исходных данных.

6.5. ПОНЯТИЕ АЛГОРИТМИЧЕСКОГО ЯЗЫКА

Достаточно распространенным способом представления алгоритма является его запись на алгоритмическом **языке**, представляющем в общем случае систему обозначений и правил для единообразной и точной записи алгоритмов и исполнения их. Отметим, что между понятиями «алгоритмический язык» и «языки программирования» есть различие; прежде всего, под исполнителем в алгоритмическом языке может подразумеваться не только компьютер, но и устройство для работы «в обстановке». Программа, записанная на алгоритмическом языке, не обязательно предназначена компьютеру. Практическая же реализация алгоритмического языка - отдельный вопрос в каждом конкретном случае.

Как и каждый язык, алгоритмический язык имеет свой словарь. Основу этого словаря составляют слова, употребляемые для записи команд, входящих в систему команд исполнителя того или иного алгоритма. Такие команды называют простыми командами. В алгоритмическом языке используют слова, смысл и способ употребления которых задан раз и навсегда. Эти слова называют служебными. Использование служебных слов делает запись алгоритма более наглядной, а форму представления различных алгоритмов - единообразной.

Алгоритм, записанный на алгоритмическом языке, должен иметь название. Название желательно выбирать так, чтобы было ясно, решение какой задачи описывает данный алгоритм. Для выделения названия алгоритма перед ним записывают служебное слово АЛГ (АЛГоритм). За названием алгоритма (обычно с новой строки) записывают его команды. Для указания начала и конца алгоритма его команды заключают в пару служебных слов НАЧ (НАЧало) и КОН (КОНец). Команды записывают последовательно.

Последовательность записи алгоритма:

АЛГ название алгоритма

НАЧ

серия команд алгоритма

КОН

Например, алгоритм, определяющий движение исполнителя-робота, может иметь вид:

АЛГ в_склад

НАЧ

вперед

поворот на 90° направо

вперед

КОН

При построении новых алгоритмов могут использоваться алгоритмы, составленные ранее. Алгоритмы, целиком используемые в составе других алгоритмов, называют вспомогательными алгоритмами. Вспомогательным может оказаться любой алгоритм из числа ранее составленных. Не исключается также, что вспомогательным в определенной ситуации может оказаться алгоритм, сам содержащий ссылку на вспомогательные алгоритмы.

Очень часто при составлении алгоритмов возникает необходимость использования в качестве вспомогательного одного и того же алгоритма, который к тому же может быть весьма сложным и громоздким. Было бы нерационально, начиная работу, каждый раз заново составлять и запоминать такой алгоритм для его последующего использования. Поэтому в практике широко используют, так называемые, встроенные (или стандартные) вспомогательные алгоритмы, т.е. такие алгоритмы, которые постоянно имеются в распоряжении исполнителя. Обращение к таким алгоритмам осуществляется так же, как и к «обычным» вспомогательным алгоритмам. У исполнителя-робота встроенным вспомогательным алгоритмом может быть перемещение в склад из любой точки рабочего поля; у исполнителя-язык программирования Бейсик это, например, встроенный алгоритм «SIN».

Алгоритм может содержать обращение к самому себе как вспомогательному и в этом случае его называют **рекурсивным**. Если команда обращения алгоритма к самому себе находится в самом алгоритме, то такую рекурсию называют прямой. Возможны случаи, когда рекурсивный вызов данного алгоритма происходит из вспомогательного алгоритма, к которому в данном алгоритме имеется обращение. Такая рекурсия называется **косвенной**. Пример прямой рекурсии:

```
АЛГ движение
НАЧ
    вперед
    вперед
    вправо
    движение
КОН
```

Алгоритмы, при исполнении которых порядок следования команд определяется в зависимости от результатов проверки некоторых условий, называют **разветвляющимися**. Для их описания в алгоритмическом языке используют специальную составную команду - команда **ветвления**. Она соответствует блок-схеме «альтернатива» и также может иметь полную или сокращенную форму. Применительно к исполнителю-роботу условием может быть проверка нахождения робота у края рабочего поля (край/не_край); проверка наличия объекта в текущей клетке (есть/нет) и некоторые другие:

ЕСЛИ условие	ЕСЛИ условие	ЕСЛИ край
ТО серия 1	ТО серия	ТО вправо
ИНАЧЕ серия2	ВСЕ	ИНАЧЕ вперед
ВСЕ		ВСЕ

Ниже приводится запись на алгоритмическом языке команды выбора (см. рис. 1.14, б), являющейся развитием команды ветвления:

```
ВЫБОР
    ПРИ условие 1:      серия 1
    ПРИ условие 2:      серия 2
    ...
    ПРИ условие N:      серия N
    ИНАЧЕ                серия N+1
ВСЕ
```

Алгоритмы, при исполнении которых отдельные команды или серии команд выполняются

неоднократно, называют циклическими. Для организации циклических алгоритмов в алгоритмическом языке используют специальную составную команду цикла. Она соответствует блок-схемам типа «итерация» и может принимать следующий вид:

ПОКА условие	НЦ	серия
НЦ		ДО условие
Серия		
КЦ	КЦ	

В случае составления алгоритмов работы с величинами можно рассмотреть и другие возможные алгоритмические конструкции, например, цикл с параметром или выбор. Подробно эти конструкции будут рассматриваться при знакомстве с реальными языками программирования.

В заключение данного параграфа приведем алгоритм, составленный для исполнителя-робота, по которому робот переносит все объекты со склада в левый нижний угол рабочего поля (поле может иметь произвольные размеры):

АЛГ перенос	АЛГ в_угол3	АЛГ до_края
НАЧ	НАЧ	НАЧ
в_угол3	до_края	ПОКА не_край
ЕСЛИ есть	вправо	НЦ
ТО	до_края	вперед
взять	вправо	КЦ
в_угол3	КОН	КОН
установить		
перенос		
ИНАЧЕ		
в_угол3		
ВСЕ		
КОН		

Контрольные вопросы

1. Каковы возможные подходы к определению понятия алгоритм?
2. Кто (что) может быть исполнителем алгоритма?
3. В чем особенности графического способа представления алгоритмов?
4. Каковы основные алгоритмические структуры?
5. Чем определяются свойства алгоритмов «дискретность», «определенность», «понятность», «результативность», «массовость»?
6. Что такое алгоритмический язык?

§7. ФОРМАЛИЗАЦИЯ ПОНЯТИЯ «АЛГОРИТМ»

7.1. ПОСТАНОВКА ПРОБЛЕМЫ

Понятие алгоритма, введенное в предыдущем параграфе, можно назвать понятием алгоритма в интуитивном смысле. Оно имеет нечеткий, неформальный характер, ссылается на некоторые точно не определенные, но интуитивно понятные вещи. Например, при определении и обсуждении свойств алгоритма мы исходили из возможностей некоторого исполнителя алгоритма. Его наличие предполагалось, но ничего определенного о нем не было известно. Говоря языком математики, ни аксиоматического, ни исчерпывающего конструктивного определения исполнителя мы так и не дали.

Установленные в предыдущем параграфе свойства алгоритмов следует называть эмпирическими. Они выявлены на основе обобщения свойств алгоритмов различной природы и имеют прикладной характер. Этих свойств достаточно для практического программирования, для создания обширного круга программ для компьютеров, станков с ЧПУ, промышленных роботов и т.д. Однако, как фундаментальное научное понятие алгоритм требует более обстоятельного изучения.

Оно невозможно без уточнения понятия «алгоритм», более строгого его описания или, как еще говорят, без его **формализации**.

Известно несколько подходов к формализации понятия «алгоритм»:

- теория конечных и бесконечных автоматов;
- теория вычислимых (рекурсивных) функций;
- λ -исчисление Черча.

Все эти возникшие исторически независимо друг от друга подходы оказались впоследствии эквивалентными. Главная цель формализации понятия алгоритма такова: подойти к решению проблемы алгоритмической разрешимости различных математических задач, т.е. ответить на вопрос, может ли быть построен алгоритм, приводящий к решению задачи. Мы рассмотрим постановку этой проблемы и некоторые результаты теории алгоритмической разрешимости задач, но вначале обсудим формализацию понятия алгоритма в теории автоматов на примере машин Поста, Тьюринга, а также нормальных алгоритмов Маркова, а затем - основы теории рекурсивных функций. Идеи λ -исчислений Черча реализованы в языке программирования LISP (глава 3).

Вместе с тем, формально определенный любым из известных способов алгоритм не может в практическом программировании заменить то, что мы называли алгоритмами в предыдущем параграфе. Основная причина состоит в том, что формальное определение резко сужает круг рассматриваемых задач, делая многие практически важные задачи недоступными для рассмотрения.

7.2. МАШИНА ПОСТА

Абстрактные (т.е. существующие не реально, а лишь в воображении) машины Поста и Тьюринга, предназначенные для доказательств различных утверждений о свойствах программ для них, были предложены независимо друг от друга (и практически одновременно) в 1936 г. американским математиком Эмилем Постом и английским математиком Алланом Тьюрингом. Эти машины представляют собой универсальных исполнителей, являющихся полностью детерминированными, позволяющих «вводить» начальные данные, и после выполнения программ «читать» результат. Машина Поста менее популярна, хотя она значительно проще машины Тьюринга. С ее помощью можно вести обучение первым навыкам составления программ для ЭВМ.

Абстрактная машина Поста представляет собой бесконечную ленту, разделенную на одинаковые клетки, каждая из которых может быть либо пустой, либо заполненной меткой «V», и головки, которая может перемещаться вдоль ленты на одну клетку вправо или влево, наносить в клетку ленты метку, если этой метки там ранее не было, стирать метку, если она была, или проверять наличие в клетке метки. Информация о заполненных метками клетках ленты характеризует состояние ленты, которое может меняться в процессе работы машины. В каждый момент времени головка («-») находится над одной из клеток ленты и, как говорят, обзорекает ее. Информация о местоположении головки вместе с состоянием ленты характеризует состояние машины Поста, рис. 1.16.

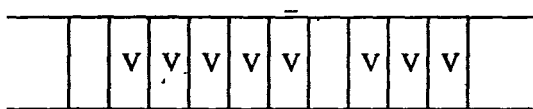


Рис. 1.16. Абстрактная машина Поста

Команда машины Поста имеет следующую структуру:

$n \ K m,$

где n - порядковый номер команды, K -действие, выполняемое головкой, m - номер следующей команды, подлежащей выполнению.

Существует всего шесть команд машины Поста, рис. 1.17:

Команда	Состояние ленты	
	до команды	после команды
Движение головки на одну клетку вправо		
Движение головки на одну клетку влево		
Нанесение метки в клетку, над которой находится головка		
Стирание метки из клетки, над которой находится головка		
Проверка наличия метки в клетке, над которой находится головка; если метка отсутствует, управление передается команде m2		
Остановка машины		

Рис. 1.17. Команды машины Поста

Ситуации, в которых головка должна наносить метку там, где она уже имеется, или, наоборот, стирать метку там, где ее нет, являются аварийными (недопустимыми).

Программой для машины Поста будем называть непустой список команд, такой что 1) на n -м месте команда с номером n ; 2) номер m каждой команды совпадает с номером какой-либо команды списка.

С точки зрения свойств алгоритмов, изучаемых с помощью машины Поста, наибольший интерес представляют причины останова машины при выполнении программы:

1) останов по команде «стоп»; такой останов называется результативным и указывает на корректность алгоритма (программы);

2) останов при выполнении недопустимой команды; в этом случае останов называется безрезультативным;

3) машина не останавливается никогда; в этом и в предыдущем случае мы имеем дело с некорректным алгоритмом (программой).

Будем понимать под начальным состоянием головки против пустой клетки левее самой левой метки на ленте.

Рассмотрим реализацию некоторых типичных элементов программ машины Поста.

1. Пусть задано исходное состояние головки и требуется на пустой ленте написать две метки: одну в секцию под головкой, вторую справа от нее. Это можно сделать по следующей программе (справа от команды показан результат ее выполнения):

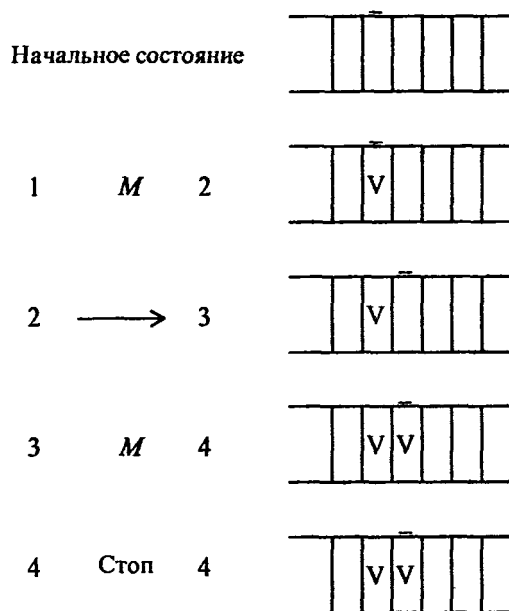


Рис. 1.18. Пример элемента программы машины Поста

2. Покажем, как можно воспользоваться командой условного перехода для организации циклического процесса. Пусть на ленте имеется запись из нескольких меток подряд и головка находится над самой крайней меткой справа. Требуется перевести головку влево до первой пустой позиции.

Программа будет иметь следующий вид:

1 $\longleftarrow$ 2

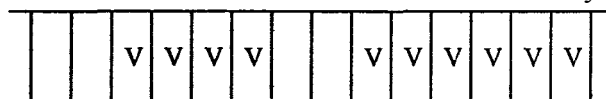
2 $\begin{cases} \longrightarrow 3 \\ \longrightarrow 1 \end{cases}$

3 Стоп 3

Команда условного перехода является одним из основных средств организации циклических процессов, например, для нахождения первой метки справа (или слева) от головки, расположенной над пустой клеткой; нахождение слева (или справа) от головки пустой клетки, если она расположена над меткой и т.д.

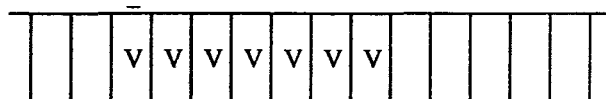
3. Остановимся на представлении чисел на ленте машины Поста и выполнении операций над ними.

Число k представляется на ленте машины Поста идущими подряд $k + 1$ метками (одна метка означает число «0»). Между двумя числами делается интервал как минимум из одной пустой секции на ленте. Например, запись чисел 3 и 5 на ленте машины Поста будет выглядеть так:



Обратим внимание, что используемая в машине Поста система записи чисел является непозиционной.

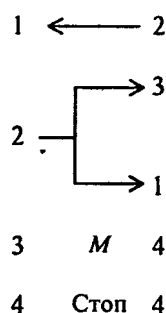
Составим программу для прибавления к произвольному числу единицы. Предположим, что на ленте записано только одно число и головка находится над одной из клеток, в которой находится метка, принадлежащая этому числу:



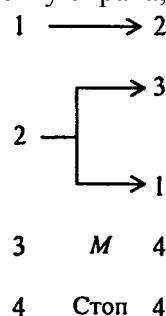
Для решения задачи можно переместить головку влево (или вправо) до первой пустой клет-

ки, а затем нанести метку.

Программа, добавляющая к числу метку слева, имеет вид:

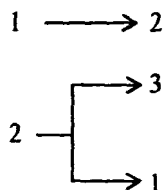


Программа, добавляющая к числу метку справа, имеет вид:

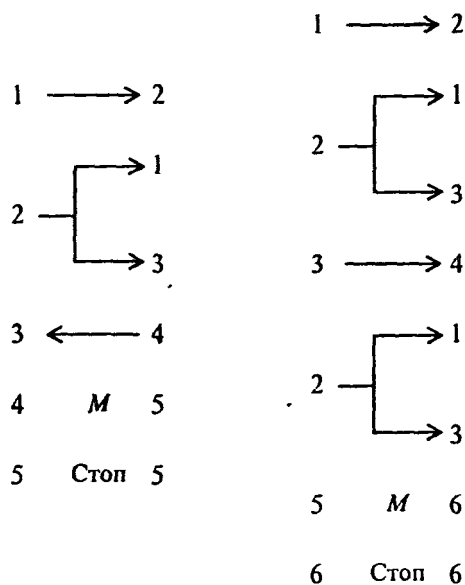


(отличие только в направлении движения головки в первой команде. Проверьте работоспособность этих программ на каких-либо частных примерах).

Предположим, что головка расположена на расстоянии нескольких клеток слева от числа, к которому нужно прибавить единицу. В этом случае программа усложняется. Появится «блок поиска числа» - две команды, приводящие головку в состояние, рассмотренное в предыдущем примере:

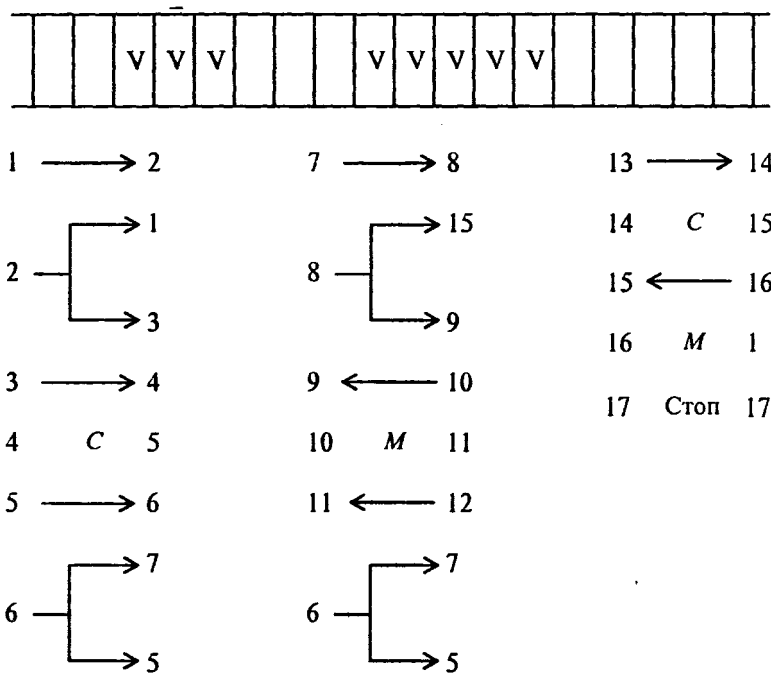


Ниже - полные тексты программ, добавляющие единицу слева и справа, соответственно:

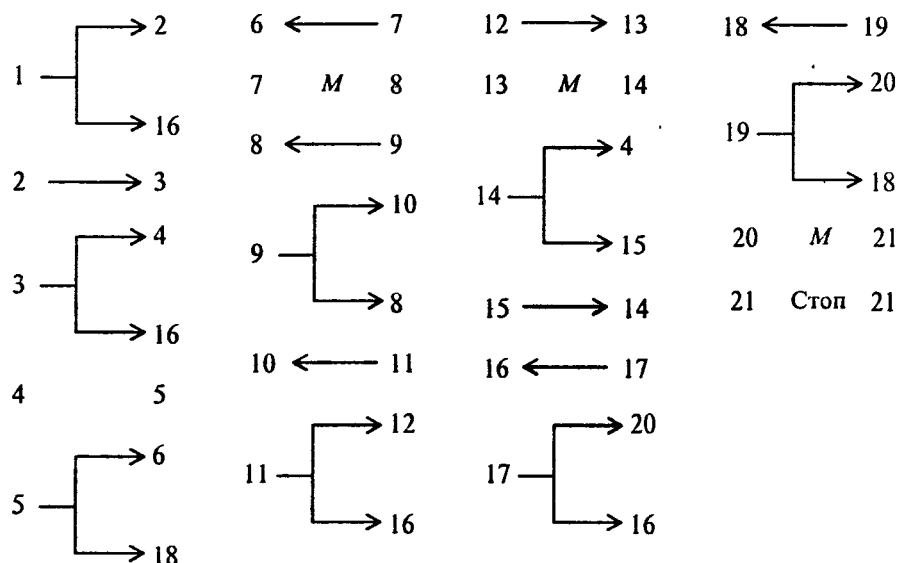


В первом случае не нужно перемещать головку к крайней левой метке числа

4. Приведем программу для сложения целых неотрицательных чисел a и b на машине Поста, когда головка находится над числом a , а число b находится правее числа a на некоторое число клеток. Эта программа реализует следующий алгоритм: первое число постепенно придвигается ко второму до их слияния, а потом стирается одна метка (иначе результат оказался бы на единицу больше правильного).



В случае более сложных начальных условий, когда неизвестно, справа или слева от головки (и на какое число клеток) находится число, можно применить такой принцип поиска числа: двигая головку вправо и влево и отмечая метками степень удаления головки от исходного положения, найти число, а потом уже применить известную программу сложения. При этом проверяется, находится ли головка над одной из меток числа и если да, то задача решена. Иначе проверяется, пуста ли секция справа от головки и следующая за ней; если обе пусты, то делается возврат головки на один шаг и ставится метка, а затем такая же операция выполняется слева и по отмеченной дорожке головка возвращается вправо и т.д. до тех пор, пока головка не натолкнется на число, после чего можно применить ранее рассмотренные выше программы:



Машину Поста можно рассматривать как упрощенную модель ЭВМ. В самом деле, как ЭВМ, так и машина Поста имеют:

- неделимые носители информации (клетки - биты), которые могут быть заполненными или незаполненными;
- ограниченный набор элементарных действий - команд, каждая из которых выполняется за один такт (шаг).

Обе машины работают на основе программы. Однако, в машине Поста информация располагается линейно и читается подряд, а в ЭВМ можно читать информацию по адресу; набор команд ЭВМ значительно шире и выразительнее, чем команды машины Поста и т.д.

73. МАШИНА ТЬЮРИНГА

Машина Тьюринга подобна машине Поста, но функционирует несколько иначе.

Машина Тьюринга (МТ) состоит из счетной ленты (разделенной на ячейки и ограниченной слева, но не справа), читающей и пишущей головки, лентопротяжного механизма и операционного исполнительного устройства, которое может находиться в одном из дискретных состояний $q_0, q_1, \dots, q_s$, принадлежащих некоторой конечной совокупности (алфавиту внутренних состояний). При этом q_0 называется начальным состоянием.

Читающая и пишущая головка может читать буквы рабочего алфавита $A = [a_0, a_1, \dots, a_t]$, стирать их и печатать. Каждая ячейка ленты в каждый момент времени занята буквой из множества A . Чаще всего встречается буква a_0 - «пробел». Головка находится в каждый момент времени над некоторой ячейкой ленты - текущей рабочей ячейкой. Лентопротяжный механизм может перемещать ленту так, что головка оказывается над соседней ячейкой ленты. При этом возможна ситуация выхода за левый край ленты (ЛК), которая является аварийной (недопустимой), или машинного останова (МО), когда машина выполняет предписание об остановке.

Порядок работы МТ (с рабочим алфавитом $a_0, a_1, \dots, a_t$ и состояниями $q_0, q_1, \dots, q_s$) описывается таблицей машины Тьюринга. Эта таблица является матрицей с четырьмя столбцами и $(s + 1)(t + 1)$ строками. Каждая строка имеет вид

$$q_i a_j v_{ij} q_{ij}, \quad 0 \leq i \leq s, \quad 0 \leq j \leq t, \quad q_{ij} \in \{q_0, q_1, \dots, q_s\}.$$

Здесь через v_{ij} обозначен элемент объединения алфавита $\{a_0, a_1, \dots, a_t\}$ и множества предписаний для лентопротяжного механизма: l - переместить ленту влево, r - переместить ленту вправо, s - остановить машину; v_{ij} - действие МТ, состоящее либо в занесении в ячейку ленты символа алфавита $a_0, a_1, \dots, a_t$, либо в движении головки, либо в останове машины; q_{ij} является последующим состоянием.

МТ работает согласно следующим правилам: если МТ находится в состоянии q_i , головка прочитывает символ a_j в рабочей ячейке. Пусть строка $q_i a_j v_{ij} q_{ij}$, начинающаяся с символов q_i, a_j , встречается только один раз в таблице. Если v_{ij} - буква рабочего алфавита, то головка стирает содержимое рабочей ячейки и заносит туда эту букву. Если v_{ij} - команда r или l для лентопротяжного механизма, то лента сдвигается на одну ячейку вправо или влево (если не происходит выход за левый край ленты) соответственно. Если $v_{ij} = s$, то происходит машинный останов.

Машина Тьюринга начинает работу из некоторой начальной конфигурации, включающей в себя начальное состояние (обычно q_0) и положение считывающе-записывающей головки над определенной ячейкой ленты, содержащей один из символов рабочего алфавита A .

Отметим, что наличие разнообразных символов в рабочем алфавите МТ позволяет представлять на ленте произвольную текстовую и числовую информацию, а переходы управляющего центра МТ в различные состояния моделируют запоминание машиной Тьюринга промежуточных результатов работы. Таблица, определяющая порядок работы МТ, не является в прямом смысле слова программой (ее предписания выполняются не последовательно, одно за другим, а описывают преобразования символов некоторого текста, находящегося на ленте). Таблицу МТ часто называют схемой машины Тьюринга или попросту отождествляют с самой машиной Тьюринга, коль скоро ее устройство и принцип функционирования известны.

Рассмотрим примеры нескольких схем машины Тьюринга.

1. Алгоритм прибавления единицы к числу n в десятичной системе счисления. Дана десятичная запись числа n (т.е. представление натурального числа n в десятичной системе счисления); требуется получить десятичную запись числа $n + 1$.

Очевидно, что внешний алфавит МТ должен состоять из десяти цифр $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ и символа пробела $_{-}$. Эти цифры записывают по одной в ячейке (по ряд, без пропусков).

Оказывается достаточным иметь два внутренних состояния машины: q_1 и q_2 .

Предположим, что в начальный момент головка находится над одной из цифр числа, а машина находится в состоянии q_1 . Тогда задача может быть решена в два этапа: движения головки к цифре единиц числа (во внутреннем состоянии q_1) и замене этой цифры на единицу большую (с учетом переноса 1 в следующий разряд, если это 9, во внутреннем состоянии q_2). Соответствующая схема МТ может иметь вид

a_i	q_i	
	q_1	q_2
0	0П q_1	1С q_2
1	1П q_1	2С q_2
2	2П q_1	3С q_2
3	3П q_1	4С q_2
4	4П q_1	5С q_2
5	5П q_1	6С q_2
6	6П q_1	7С q_2
7	7П q_1	8С q_2
8	8П q_1	9С q_2
9	9П q_1	0С q_2
-	-Л q_1	1С q_2

2. Алгоритм записи числа в десятичной системе счисления.

Дана конечная последовательность меток, записанных в клетки ленты подряд, без пропусков. Требуется записать в десятичной системе число этих меток (пересчитать метки).

Суть алгоритма может состоять в том, что к числу 0, записанному на ленте в начале работы машины, машина добавляет 1, стирая метку за меткой, так что вместо нуля возникает число $0 + k$.

Легко могут быть построены алгоритмы сложения чисел, их перемножения, нахождения наибольшего общего делителя и т.д. Однако, главная цель введения машин Поста и Тьюринга не программирование для них, а изучение свойств алгоритмов и проблемы алгоритмической разрешимости задач.

В зависимости от числа используемых лент, их назначения и числа состояний устройства управления можно рассматривать различные модификации машин Тьюринга.

Предположим, мы расширили определение МТ, добавив определенное состояние q устройства управления машины. Будем говорить, что если устройство управления переходит в состояние q_0 для заданного входного слова x , то машина допускает x ; если устройство переходит в состояние q_x , то машина запрещает x . Такую машину будем называть машиной Тьюринга с двумя выходами. Могут быть рассмотрены многочисленные варианты машины Тьюринга, имеющие некоторое конечное число лент. В каждой клетке этих лент может находиться один из символов внешнего алфавита $A = \{a_0, a_1, \dots, a_n\}$. Устройство управления машиной в каждый момент времени находится в одном из конечного множества состояний $Q = \{q_0, q_1, \dots, q_m\}$. Для K -ленточной машины конфигурация ее в i -й момент времени описывается системой k -слов вида:

$$\begin{aligned} a_{i11} \dots a_{i1l} q_i a_{i1l+1} \dots s_{i1t}; \\ a_{ik1} \dots a_{ikl} q_i a_{ikl+1} \dots a_{ikv}; \end{aligned}$$

первый индекс соответствует моменту времени, второй - номеру ленты, третий - номеру клетки, считая слева направо. Говорят, что машина выполняет команду

$$q_i a_{a1} \dots a_{ak} \rightarrow q_j a_{b1} k_1 \dots a_{bk} k_k, \\ K = \{Л, С, П\}.$$

Если, находясь в состоянии q_i и обозревая ячейки с символами $a_{a1} \dots a_{ak}$, машина переходит в состояние q_j , заменяя содержимое ячеек соответственно символами $a_{b1} \dots a_{bk}$, то после этого ленты соответственно сдвигаются в направлениях $k_1 \dots k_k$.

До сих пор принималось, что различные алгоритмы осуществляются на различных машинах Тьюринга, отличающихся набором команд, внутренним и внешним алфавитами. Однако, можно построить универсальную машину Тьюринга, способную выполнять любой алгоритм любой машины Тьюринга. Это достигается путем кодирования конфигурации и программы любой данной машины Тьюринга в символах внешнего алфавита универсальной машины. Само кодирование должно выполняться следующим образом:

1) различные символы должны заменяться различными кодовыми группами, но один и тот же символ должен заменяться всюду, где бы он не встретился, одной и той же кодовой группой;

- 2) строки кодовых записей должны однозначно разбиваться на отдельные кодовые группы;
- 3) должна иметься возможность распознать кодовые группы, соответствующие командам L, P, C , различать кодовые группы, соответствующие символам внешнего алфавита и внутренним состояниям.

Для сравнения структур различных машин и оценки их сложности необходимо иметь соответствующую меру сложности машин. К.Шеннон предложил рассматривать в качестве такой меры произведение числа символов внешнего алфавита и числа внутренних состояний. Большой интерес вызывает задача построения универсальной машин Тьюринга наименьшей сложности.

Может быть рассмотрено еще одно обобщение машин Тьюринга: их композиции. Операции композиции, выполняемые над алгоритмами, позволяют образовывать новые, более сложные алгоритмы из ранее известных простых алгоритмов. Поскольку машина Тьюринга - алгоритм, то операции композиции применимы и к машинам Тьюринга. Рассмотрим основные из них: произведение, возведение в степень, итерацию.

Пусть заданы машины Тьюринга T_1 и T_2 , имеющие общий внешний алфавит $A = \{a_0, a_1, \dots, a_m\}$ и внутренние состояния $Q_1 = \{q_0, q_1, \dots, q_n\}$ и $Q_2 = \{q_0, q_1, \dots, q_l\}$ соответственно. Композицией или произведением машины T_1 и машины T_2 будем называть машину T с тем же внешним алфавитом $A = \{a_0, a_1, \dots, a_m\}$ и набором внутренних состояний $Q = \{q_0, q_1, \dots, q_2, q_{n+1}, \dots, q_{n+l}\}$ и программой, эквивалентной последовательному выполнению программ машин T_1 и T_2 :

$$T = T_1 * T_2.$$

Таким же образом определяется операция возведения в степень: n -й степенью машины T называется произведение $T \dots T$ с n сомножителями.

Операция итерации применима к одной машине и определяется следующим образом. Пусть машина T_1 имеет несколько заключительных состояний. Выберем ее r -е заключительное состояние и отождествим его в схеме машины с ее начальным состоянием. Полученная машина T является результатом итерации машины T_1 : $T = T_1$.

Прежде чем остановиться на проблеме алгоритмической разрешимости задач обратимся к другим способам формализации понятия алгоритма.

7.4. НОРМАЛЬНЫЕ АЛГОРИТМЫ МАРКОВА

Для формализации понятия алгоритма российский математик А.А.Марков предложил использовать ассоциативные исчисления.

Рассмотрим некоторые понятия ассоциативного исчисления. Пусть имеется алфавит (конечный набор различных символов). Составляющие его символы будем называть буквами. Любая конечная последовательность букв алфавита (линейный их ряд) называется словом в этом алфавите.

Рассмотрим два слова N и M в некотором алфавите A . Если N является частью M , то говорят, что N входит в M .

Зададим в некотором алфавите конечную систему подстановок $N - M, S - T, \dots$, где $N, M, S, T, \dots$ - слова в этом алфавите. Любую подстановку $N-M$ можно применить к некоторому слову K следующим способом: если в K имеется одно или несколько вхождений слова N , то любое из них может быть заменено словом M , и, наоборот, если имеется вхождение M , то его можно заменить словом N .

Например, в алфавите $A = \{a, b, c\}$ имеются слова $N = ab, M = bcb, K = abcbcbab$. Заменяя в слове K слово N на M , получим $bcbcbcbab$ или $abcbcbcb$, и, наоборот, заменив M на N , получим $aabcbab$ или $abcaabab$.

Подстановка $ab - bcb$ недопустима к слову $bacb$, так как ни ab , ни bcb не входит в это слово. К полученным с помощью допустимых подстановок словам можно снова применить допустимые подстановки и т.д. Совокупность всех слов в данном алфавите вместе с системой допустимых подстановок называют **ассоциативным исчислением**. Чтобы задать ассоциативное исчисление, достаточно задать алфавит и систему подстановок.

Слова P_1 и P_2 в некотором ассоциативном исчислении называются смежными, если одно из них может быть преобразовано в другое однократным применением допустимой подстановки.

Последовательность слов $P, P_1, P_2, \dots, M$ называется дедуктивной цепочкой, ведущей от слова P к слову M , если каждое из двух рядом стоящих слов этой цепочки - смежное.

Слова P и M называют эквивалентными, если существует цепочка от P к M и обратно.

Пример

Алфавит	Подстановки
$\{a, b, c, d, e\}$	$ac - ca,$
	$ad - da;$
	$bc - cb;$
	$bd - db;$
	$abac - abace$
	$eca - ae$
	$eda - be$
	$edb - be$

Слова $abcde$ и $acbde$ - смежные (подстановка $bc - cb$). Слова $abcde - cadbe$ эквивалентны.

Может быть рассмотрен специальный вид ассоциативного исчисления, в котором подстановки являются ориентированными: $N \rightarrow M$ (стрелка означает, что подстановку разрешается производить лишь слева направо). Для каждого ассоциативного исчисления существует задача: для любых двух слов определить, являются ли они эквивалентными или нет.

Любой процесс вывода формул, математические выкладки и преобразования также являются дедуктивными цепочками в некотором ассоциативном исчислении. Построение ассоциативных исчислений является универсальным методом детерминированной переработки информации и позволяет формализовать понятие алгоритма.

Введем понятие алгоритма на основе ассоциативного исчисления: алгоритмом в алфавите A называется понятное точное предписание, определяющее процесс над словами из A и допускающее любое слово в качестве исходного. Алгоритм в алфавите A задается в виде системы допустимых подстановок, дополненной точным предписанием о том, в каком порядке нужно применять допустимые подстановки и когда наступает остановка.

Пример

Алфавит:	Система подстановок B :
$A = \{a, b, c\}$	$cb - cc$
	$cca - ab$
	$ab - bca$

Предписание о применении подстановок: в произвольном слове P надо сделать возможные подстановки, заменив левую часть подстановок на правую; повторить процесс с вновь полученным словом.

Так, применяя систему подстановок B из рассмотренного примера к словам $babaac$ и $bcacabc$ получаем:

$babaac \rightarrow bbcaaac \rightarrow \text{остановка}$

$bcacabc \rightarrow bcacbcac \rightarrow bcacccac \rightarrow bcacabc \rightarrow \text{бесконечный процесс (остановки нет), так как мы получили исходное слово.}$

Предложенный А.А.Марковым способ уточнения понятия алгоритма основан на понятии нормального алгоритма, который определяется следующим образом. Пусть задан алфавит A и система подстановок B . Для произвольного слова P подстановки из B подбираются в том же порядке, в каком они следуют в B . Если подходящей подстановки нет, то процесс останавливается. В противном случае берется первая из подходящих подстановок и производится замена ее правой частью первого вхождения ее левой части в P . Затем все действия повторяются для получившегося слова P_1 . Если применяется последняя подстановка из системы B , процесс останавливается.

Такой набор предписаний вместе с алфавитом A и набором подстановок B определяют нормальный алгоритм. Процесс останавливается только в двух случаях: 1) когда подходящая подстановка не найдена; 2) когда применена последняя подстановка из их набора. Различные нормальные алгоритмы отличаются друг от друга алфавитами и системами подстановок.

Приведем пример нормального алгоритма, описывающего сложение -натуральных чисел (представленных наборами единиц).

Пример

Алфавит:
 $A = (+, 1)$

Система подстановок B :
 $1 + \rightarrow + 1$
 $+ 1 \rightarrow 1$
 $1 \rightarrow 1$

Слово P : $11+11+111$

Последовательная переработка слова P с помощью нормального алгоритма Маркова проходит через следующие этапы:

$P = 11 + 11 + 111$
 $P_1 = 1 + 111 + 111$
 $P_2 = + 1111 + 111$
 $P_3 = + 111 + 1111$
 $P_4 = + 11 + 11111$

$P_5 = + 1 + 111111$
 $P_6 = ++ 1111111$
 $P_7 = + 1111111$
 $P_8 = 1111111$
 $P_9 = 1111111$

Нормальный алгоритм Маркова можно рассматривать как универсальную форму задания любого алгоритма. Универсальность нормальных алгоритмов декларируется принципом нормализации: для любого алгоритма в произвольном конечном алфавите A можно построить эквивалентный ему нормальный алгоритм над алфавитом A .

Разъясним последнее утверждение. В некоторых случаях не удастся построить нормальный алгоритм, эквивалентный данному в алфавите A , если использовать в подстановках алгоритма только буквы этого алфавита. Однако, можно построить требуемый нормальный алгоритм, производя расширение алфавита A (добавляя к нему некоторое число новых букв). В этом случае говорят, что построенный алгоритм является алгоритмом над алфавитом A , хотя он будет применяться лишь к словам в исходном алфавите A .

Если алгоритм N задан в некотором расширении алфавита A , то говорят, что N есть нормальный алгоритм над алфавитом A .

Условимся называть тот или иной алгоритм нормализуемым, если можно построить эквивалентный ему нормальный алгоритм, и ненормализуемым в противном случае. Принцип нормализации теперь может быть высказан в видоизмененной форме: все алгоритмы нормализуемы.

| Данный принцип не может быть строго доказан, поскольку понятие произвольного алгоритма не является строго определенным и основывается на том, что все Известные в настоящее время алгоритмы являются нормализуемыми, а способы композиции алгоритмов, позволяющие строить новые алгоритмы из уже известных, не выводят за пределы класса нормализуемых алгоритмов. Ниже перечислены способы композиции нормальных алгоритмов.

I. *Суперпозиция алгоритмов.* При суперпозиции двух алгоритмов A и B выходное слово первого алгоритма рассматривается как входное слово второго алгоритма B . Результат суперпозиции C может быть представлен в виде $C(p) = B(A(p))$.

II. *Объединение алгоритмов.* Объединением алгоритмов A и B в одном и том же алфавите называется алгоритм C в том же алфавите, преобразующий любое слово p , содержащееся в пересечении областей определения алгоритмов A и B , в записанные рядом слова $A(p)$ и $B(p)$.

III. *Разветвление алгоритмов.* Разветвление алгоритмов представляет собой композицию D трех алгоритмов A , B и C , причем область определения алгоритма D является пересечением областей определения всех трех алгоритмов A , B и C , а для любого слова p из этого пересечения $D(p) = A(p)$, если $C(p) = e$, $D(p) = B(p)$, если $C(p) = e$, где e - пустая строка.

IV. *Итерация алгоритмов.* Итерация (повторение) представляет собой такую композицию C двух алгоритмов A и B , что для любого входного слова p соответствующее слово $C(p)$ получается в результате последовательного многократного применения алгоритма A до тех пор, пока не получится слово, преобразуемое алгоритмом B .

Нормальные алгоритмы Маркова являются не только средством теоретических построений, но и основой специализированного языка программирования, применяемого как язык символьных преобразований при разработке систем искусственного интеллекта. Это один из немногих языков, разработанных в России и получивших известность во всем мире.

Существует строгое доказательство того, что по возможностям преобразования нормальные алгоритмы Маркова эквивалентны машинам Тьюринга.

7.5. РЕКУРСИВНЫЕ ФУНКЦИИ

Еще одним подходом к проблеме формализации понятия алгоритма являются, так называемые, рекурсивные функции. Исторически этот подход возник первым, поэтому в математических исследованиях, посвященных алгоритмам, он имеет наибольшее распространение.

Рекурсией называется способ задания функции, при котором значение функции при определенном значении аргументов выражается через уже заданные значения функции при других значениях аргументов. Применение рекурсивных функций в теории алгоритмов основано на идее нумерации слов в произвольном алфавите последовательными натуральными числами. Таким образом любой алгоритм можно свести к вычислению значений некоторой целочисленной функции при целочисленных значениях аргументов.

Введем несколько основных понятий. Пусть X, Y - два множества. Частичной функцией (или отображением) из X в Y будем называть пару $\langle D(f), f \rangle$, состоящую из подмножества $D(f) \subset X$ (называемого областью определения f) и отображения $f: D(f) \rightarrow Y$. Если $D(f)$ пусто, то f нигде не определена. Будем считать, что существует единственная нигде не определенная частичная функция.

Через N будем обозначать множество натуральных чисел. Через $(N)^n$ (при $n \geq 1$) будем обозначать n -кратное декартово произведение N на себя, т.е. множество упорядоченных n -ок $(x_1, \dots, x_n)$, $x_i \in N$. Основным объектом дальнейших построений будут частичные функции из $(N)^m$ в $(N)^n$ для различных m и n .

Частичная функция f из $(N)^m$ в $(N)^n$ называется вычислимой, если можно указать такой алгоритм («программу»), который для входного набора $x \in (N)^m$ дает на выходе $f(x)$, если $x \in D(f)$ и нуль, если $x \notin D(f)$. В этом определении неформальное понятие алгоритма (программы) оказывается связанным (отождествленным) с понятием вычислимости функции. Вместо алгоритмов далее будут изучаться свойства вычисляемых функций. Вместо вычисляемых функций оказывается необходимым использовать более широкий класс функций (и более слабое определение) - полувывчисляемые функции. Частичная функция из $(N)^m$ в $(N)^n$ полувывчислима, если можно указать такой алгоритм (программу), который для входного набора $x \in (N)^m$ дает на выходе $x \in D(f)$, или алгоритм работает неопределенно долго, если $x \notin D(f)$. Очевидно, что вычисляемые функции полувывчислимы, а всюду определенные полувывчисляемые функции вычислимы.

Частичная функция f называется невычислимой, если она не является ни вычислимой, ни полувывчисляемой.

Из вновь введенных понятий основным является полувывчислимость, так как вычислимость сводится к нему. Существуют как невычисляемые функции, так и функции, являющиеся полувывчислимыми, но не вычислимыми. Пример такой функции:

$$g(t) = \begin{cases} 1, & \text{если уравнение } P(t, x_1, \dots, x_n) \text{ разрешимо,} \\ & \text{не определена в противном случае.} \end{cases}$$

Можно показать, что существует такой многочлен с целыми коэффициентами $P(t, x_1, \dots, x_n)$, что $g(t)$ - невычислима. Однако, легко видеть, что $g(t)$ - полувывчислима.

Фундаментальным открытием теории вычислимости явился, так называемый, тезис Черча, который в слабой форме имеет следующий вид: можно явно указать а) семейство простейших полувывчисляемых функций; б) семейство элементарных операций, которые позволяют строить по одним полувывчисляемым функциям другие полувывчисляемые функции с тем свойством, что любая полувывчисляемая функция получается за конечное число шагов, каждый из которых состоит в применении одной из элементарных операций к ранее построенным или к простейшим функциям.

Простейшие функции:

$\text{suc}: N \rightarrow N$; $\text{suc}(x) = x+1$ - определение следующего за x числа;

$I^{(n)}: (N)^n \rightarrow N$; $I^{(n)}(x_1, \dots, x_n) = 1$, $n \geq 0$ - определение «размерности» области определения функции;

$pr_i^n: (N)^n \rightarrow N$; $pr_i^n(x_1, \dots, x_n) = x_i$, $x \geq 1$ - «проекция» области определения на одну из переменных.

Элементарные операции над частичными функциями:

а) *композиция* (или подстановка) ставит в соответствие паре функций f из $(N)^m$ в $(N)^n$ и g из $(N)^n$ в $(N)^p$ функцию $h = g \circ f$ из $(N)^m$ в $(N)^p$, которая определяется как

$$D(g \circ f) = f^{-1}(D(g)) = \{x(N)^m \mid xD(f), f(x)D(g)\} \quad (g \circ f)(x) = g(f(x));$$

б) *соединение* ставит в соответствие частичным функциям f_i из $(N)^{n_i}$, $i = 1, \dots, k$ функцию $(f_1, \dots, f_k)$ из $(N)^{n_1} \times \dots \times (N)^{n_k}$, которая определяется как

$$D(f_1, \dots, f_k) = D(f_1) \cap \dots \cap D(f_k), \\ (f_1, \dots, f_k)(x_1, \dots, x_m) = (f_1(x_1, \dots, x_m), \dots, f_k(x_1, \dots, x_m));$$

в) *рекурсия* ставит в соответствие паре функций f из $(N)^n$ в N и g из $(N)^{n+2}$ в N функцию h из $(N)^{n+2}$ в N , которая определяется рекурсией по последнему аргументу

$$h(x_1, \dots, x_n, 1) = f(x_1, \dots, x_n) \text{ (начальное условие)}, \\ h(x_1, \dots, x_n, k+1) = g(x_1, \dots, x_n, k, h(x_1, \dots, x_n, k)) \text{ при } k \geq 1 \text{ (рекурсивный шаг)}.$$

Область определения $D(h)$ описывается также рекурсивно:

$$\langle x_1, \dots, x_n, 1 \rangle D(h) \leftrightarrow \langle x_1, \dots, x_n \rangle D(f); \\ \langle x_1, \dots, x_n, k+1 \rangle D(h) \leftrightarrow \langle x_1, \dots, x_n, k \rangle D(h); \\ \langle x_1, \dots, x_n, k, h(x_1, \dots, x_n, k) \rangle D(g) \text{ при } k \geq 1;$$

г) *операция m* , которая ставит в соответствие частичной функции f из $(N)^{n+1}$ в N частичную функцию h из $(N)^n$ в N , которая определяется как

$$D(h) = \{\langle x_1, \dots, x_n, k \rangle \mid \exists x_{n+1} \geq 1, \\ f(x_1, \dots, x_n, x_{n+1}) = 1 \text{ и } \langle x_1, \dots, x_n, k \rangle D(f) \text{ для всех } k \leq x_{n+1}\}, \\ h(x_1, \dots, x_n) = \min\{x_{n+1} \mid f(x_1, \dots, x_n, x_{n+1}) = 1\}.$$

Операция m позволяет вводить в вычисления перебор объектов для отыскания нужного в бесконечном семействе.

Теперь, когда введены простейшие функции и элементарные операции, можно дать следующие основные определения:

а) последовательность частичных функций $f_1, \dots, f_N$ называют частично рекурсивным (соответственно примитивно рекурсивным) описанием функции $f_N = f$, если f_1 - одна из простейших функций; f_i для всех $i \geq 2$ либо является простейшей функцией, либо получается применением одной из элементарных операций к некоторым из функций $f_1, \dots, f_{i-1}$ (соответственно одной из элементарных операций, кроме m);

б) функция f называется частично рекурсивной (соответственно примитивно рекурсивной), если она допускает частично рекурсивное (соответственно примитивно рекурсивное) описание.

Теперь можно привести тезис Черча в обычной форме:

а) функция f полувычислима, если и только если она частично рекурсивна;

б) функция f вычислима, если и только если рекурсивны f и характеристическая функция $X_{D(f)}$.

Характеристическая функция подмножества X в Y ($X \subset Y$) есть такая функция, что

$$\chi(x) = \begin{cases} 1, & \text{если } x \in X, \\ 0, & \text{если } x \in Y - X. \end{cases}$$

Тезис Черча может использоваться как определение алгоритмической неразрешимости.

Пусть имеется счетная последовательность «задач» $P_1, P_2, \dots$, которые имеют ответ «да» или «нет». Такая последовательность носит название «массовой проблемы». Свяжем с ней функцию f из N в N :

$$D(f) = \{i \mid P_i \text{ имеет ответ «да»}\};$$

$$f(i) = 1, \text{ если } i \in D(f).$$

Массовая проблема P называется алгоритмически разрешимой, если функции f и $X_{D(f)}$ частично рекурсивны. В противном случае P называется алгоритмически неразрешимой.

Контрольные вопросы и задания

1. Для чего необходимо формализовать понятие алгоритма?
2. Что означает фраза: «Машины Поста и Тьюринга являются абстрактными машинами»?
3. Для чего предназначены машины Поста и Тьюринга?
4. Как «устроена» машина Поста?
5. Перечислите и запишите команды машины Поста.
6. С помощью бумаги, карандаша и стиральной резинки «исполните» вместо машины Поста программы сложения чисел из текста.
7. Составьте (и проверьте) программу для машины Поста, создающую на ленте копию заданной последовательности меток справа от нее.
8. Пользуясь предыдущей программой, составьте программу умножения чисел для машины Поста.
9. Как «устроена» машина Тьюринга?
10. Каков принцип исполнения программы машиной Тьюринга?
11. Сравните машины Поста и Тьюринга. Укажите различия.
12. Выполните вместо машины Тьюринга примеры программ из текста.
13. Каким образом могут быть обобщена машина Тьюринга?
14. Что такое ассоциативное исчисление?
15. Постройте дедуктивную цепочку от слова «мука» к слову «торт», заменяя каждый раз по одной букве так, чтобы каждый раз получалось слово.
16. Дайте определение нормального алгоритма Маркова.
17. В чем состоит принцип нормализации алгоритмов?
18. Охарактеризуйте способы композиции нормальных алгоритмов.
19. Как алгоритм может быть связан с рекурсивной функцией?
20. Дайте определения частичной, полувыхислимой и вычислимой функции.
21. В чем состоит тезис Черча в слабой и в обычной формах?
22. Перечислите простейшие функции.
23. Перечислите элементарные операции.
24. Чем отличается рекурсивная функция от примитивно-рекурсивной?
25. Дайте определение частично-рекурсивной функции.
26. Что называется массовой проблемой? Что означает алгоритмическая разрешимость массовой проблемы?

§ 8. ПРИНЦИПЫ РАЗРАБОТКИ АЛГОРИТМОВ И ПРОГРАММ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

8.1. ОПЕРАЦИОНАЛЬНЫЙ ПОДХОД

В настоящее время создание алгоритмов - написание программ для электронных вычислительных машин - стало видом человеческой деятельности. Важнейший конструктивный компонент программирования, не зависящий от особенностей синтаксиса языков программирования и специфики функционирования конкретных вычислительных машин, - разработка алгоритма.

Подходы к созданию алгоритмов и требования к ним существенно изменялись в ходе эво-

люции компьютеров. Первоначально, в эпоху ЭВМ 1 -го и 2-го поколений, когда они были еще мало распространены, машинное время было дорого, а возможности ЭВМ очень скромны (с точки зрения сегодняшних достижений), основным требованием к алгоритму была его узко понимаемая эффективность:

1) минимальные требования в отношении оперативной памяти компьютера - программа должна была использовать наименьшее возможное число ячеек оперативной памяти компьютера;

2) минимальное время исполнения (минимальное число операций). При этом программы составлялись из команд, непосредственно или почти непосредственно исполнявшихся компьютером (точнее говоря, процессором):

- операции присваивания;
- простейших арифметических операций;
- операций сравнения чисел;
- операторов безусловного и условных переходов (изменяющих порядок вычисления команд в программе);
- операторов вызова подпрограмм (вспомогательных алгоритмов).

Такой подход в программировании (создании алгоритмов), ориентированный на непосредственно выполняемые компьютером операции, можно назвать операциональным.

Рассмотрим подробнее операции, которые выполняются компьютером и которые являются шагами программы при операциональном подходе.

Операция присваивания состоит в том, что некоторое значение фигурирующей в программе величины помещается в ячейку памяти компьютера. Эта ячейка может либо принадлежать оперативной памяти, либо находиться в арифметико-логическом устройстве, выполняющем основные операции (это устройство - часть процессора). После операции присваивания указанное значение сохраняется в ячейке памяти, куда оно было помещено, пока не будет заменено другим в результате другого присваивания. Ячейка памяти, где размещается значение, в программе обозначается именем (идентификатором) соответствующей переменной. Примеры идентификаторов: a , x , y_1 , y_2 . Важно помнить, что переменные и, соответственно, их значения могут быть разных типов - числовые (целые или действительные), литерные или логические. Значения различных типов представляются (кодируются) в компьютере по-разному, поэтому они должны соответствовать типам переменных, которым они присваиваются. При разработке алгоритма следует всегда помнить и тщательно различать типы переменных.

Набор простейших арифметических операций «сложения» (+), «вычитания» (-), «умножения» (*) и «деления» (/) (причем во многих случаях следует тщательно отличать деление, выполняемое над целыми числами - в этом случае операция деления распадается на деление нацело и вычисление остатка от деления) позволяет записывать арифметические выражения с использованием числовых констант и идентификаторов переменных. Для определения порядка операций в выражениях чаще всего используют стандартное математическое соглашение о старшинстве операции, согласно которому старшими и выполняемыми в 1-ю очередь являются умножение и деление, а младшими - сложение и вычитание. Для изменения «естественного» порядка выполняемых операций служат скобки. Сравните, например, порядок операций в выражениях:

$$(a + 2) * x \text{ и } a + 2 * x.$$

Что же касается порядка выполнения операций одного старшинства, то они, как правило, выполняются в порядке записи в выражении.

Операция сравнения числовых значений фактически сводится к определению знака разности этих значений. Этот знак отображается с помощью специальной ячейки памяти (флага знака результата) вычислительного устройства компьютера и может использоваться при выполнении условных переходов между командами (шагами) алгоритма.

Чтобы понять, что такое условные и безусловные переходы при выполнении алгоритма, надо исходить из того, что шаги или команды алгоритма обладают метками или адресами, и, помимо естественного порядка выполнения команд соответственно их записи, возможен и другой порядок, при котором последовательность выполнения команд определяется переходами на команды с определенными метками или адресами. **Безусловным** называется переход, для которого изменение порядка выполнения команд раз и навсегда определено и не зависит ни от каких усло-

вий. Условным называется переход, для которого порядок выполнения команд определяется по некоторому условию, чаще всего условию сравнения величин числовых типов.

Операция вызова подпрограммы (вспомогательного алгоритма) - это такой переход в последовательности команд алгоритма, при котором на определенном этапе выполнения алгоритма происходит вначале переход на другую программу (подпрограмму по отношению к той, откуда совершен переход), а затем, после ее завершения, возврат в точку вызова подпрограммы и продолжение выполнения команд, начиная со следующей после команды вызова подпрограммы, в их естественном порядке. Очевидно, что операция вызова подпрограммы представляет собой переход, при котором запоминается адрес команды, следующей за командой вызова подпрограммы, что позволяет вернуться к исходному алгоритму (головной программе) после выполнения вспомогательного алгоритма (подпрограммы).

Отметим, что универсальность существующих компьютеров достигается за счет определенного набора команд, типа только что описанного, и автоматического механизма их выполнения, а проблема решения задачи с помощью компьютера состоит лишь в преобразовании решаемой задачи в последовательность этих команд. В качестве примера рассмотрим операционное представление алгоритма вычисления квадратного корня из положительного числа a с помощью рекуррентной формулы:

$$x_{n+1} = (x_n + a/x_n)/2,$$

$$n = 0, 1, 2, \dots$$

Можно показать, что $\lim_{n \rightarrow \infty} x_n = \sqrt{a}$.

Будем обозначать через x_0 нулевое приближение (за него в данном случае можно принять любое положительное число), через ε заданную точность вычислений и через c_0 какое-нибудь число, удовлетворяющее условию $0 < c_0 < \sqrt{a}$, необходимое для оценки достигнутой точности с помощью неравенства

$$x_n < \sqrt{a} < \frac{x_n^2 - a}{2c_0}.$$

Алгоритм вычисления $\sqrt{a}$.

- 1) ввести числа a , ε , x_0 , c_0 ;
- 2) присвоить переменной x значение y ;
- 3) присвоить переменной y значение a/x ;
- 4) присвоить переменной y значение $x + y$;
- 5) присвоить переменной x значение $y/2$;
- 6) присвоить переменной y значение x^2 ;
- 7) присвоить переменной y значение $y - a$;
- 8) присвоить переменной y значение y/c_0 ;
- 9) присвоить переменной δ значение $y/2$;
- 10) сравнить δ и ε , если $\delta > \varepsilon$, то перейти к команде 3, иначе перейти к следующей команде;
- 11) вывести числа x , a и ε ;
- 12) стоп.

В этом алгоритме все команды, кроме 10, предполагают переход к следующим за ними по записи командам, и лишь команда 10, являющаяся командой условного перехода, меняет порядок исполнения команд - после нее в нарушение порядка может выполняться команда 3, т.е. она определяет циклическую конструкцию в алгоритме.

Поясним эту программу. Команда 2 помещает значение начального приближения x_0 в ячейку памяти, в которой хранятся значения переменной x (на каждом этапе вычислений в этой ячейке хранится значение x , равное значению одного из членов рекуррентной последовательности x_n).

Команды 3-5 вычисляют по числу x число $(x + a/x)/2$. Результат помещается в ячейку памяти, в которой хранится значение переменной x , при этом старое значение «стирается» новым. Команды 6-9 вычисляют величину

$$\delta = (x^2 - a) / 2c_0,$$

с помощью которой оценивается сверху разность $x - \sqrt{a}$. Важное значение имеет команда 10. По ней не производятся вычисления, а сравниваются между собой вычисленное значение 5 и заданная точность ε . Если $\delta > \varepsilon$, то управляющее устройство вернет вычислительный процесс к команде 3 и заставит повторить процесс.

В противном случае, когда требуемая точность достигнута, печатается полученный результат и работа прекращается.

Данный алгоритм весьма экономичен: в качестве рабочих он использует всего две ячейки памяти (для переменных x и y), его команды так продуманы, что никакие операции не выполняются с избыточным повторением.

В данном примере не были использованы какие-либо специальные обозначения команд, чтобы сделать их независимыми от языка конкретных ЭВМ (такие языки называют Ассемблерами), чтобы стал ясен общий характер операционального подхода к разработке алгоритмов. Однако, ориентированность этого подхода на возможности и особенности ЭВМ с появлением большого числа компьютеров 3-го и особенно 4-го поколений не позволяла перейти к массовому промышленному программированию и стала сдерживать развитие вычислительной техники. Отметим основные недостатки алгоритмов, к которым приводил операциональный подход:

- злоупотребление командой условного и безусловного переходов зачастую приводило к очень запутанной структуре программы, напоминавшей по образному сравнению «блюдо спагетти»;
- вместе с разнообразными уловками, направленными на повышение эффективности программы (т.е. минимальных требований к оперативной памяти и минимального времени выполнения), это приводило к непонятности программ, их ненадежности, трудностям в отладке и модификации, делая программирование трудоемким, сложным и чрезвычайно дорогостоящим.

Необходимость ориентироваться на ограниченный набор команд компьютера, на его скромные возможности приводила к огромной трудоемкости, к сложности программ, к проблемам, связанным с ошибками в них. В результате узким местом в развитии вычислительной техники оказалось именно программирование.

8.2. СТРУКТУРНЫЙ ПОДХОД

С появлением массовых ЭВМ 3-го поколения устаревшая технология программирования оказалась основным фактором, сдерживающим развитие и распространение компьютерных (информационных) технологий, что подтолкнуло ведущие в этой сфере деятельности фирмы, в первую очередь IBM, к разработке новых методологий программирования. Появившийся в начале 1970-х годов новый подход к разработке алгоритмов получил название структурного.

С появлением структурного программирования описанные выше трудности были во многом преодолены. В основе технологических принципов структурного программирования лежит утверждение о том, что логическая структура программы может быть выражена комбинацией трех базовых структур: следования, ветвления и цикла (это содержание теоремы Бема-Якопини).

Следование - самая важная из структур. Она означает, что действия могут быть выполнены друг за другом, рис. 1.19:

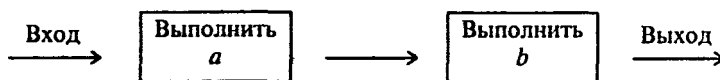


Рис. 1.19. Структура «следование»

Эти прямоугольники могут представлять как одну единственную команду, так и множество операторов, необходимых для выполнения сложной обработки данных.

Ветвление - это структура, обеспечивающая выбор между двумя альтернативами. Выполняется проверка, а затем выбирается один из путей (рис. 1.20).

Эта структура называется также «ЕСЛИ - ТО - ИНАЧЕ», или «развилка». Каждый из путей

(ТО или ИНАЧЕ) ведет к общей точке слияния, так что выполнение программы продолжается независимо от того, какой путь был выбран.

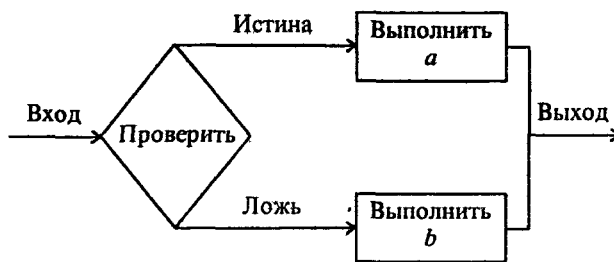


Рис. 1.20. Структура «ветвление»

Может оказаться, что для одного из результатов проверки ничего предпринимать не надо. В этом случае можно применять только один обрабатывающий блок, рис.1.21:

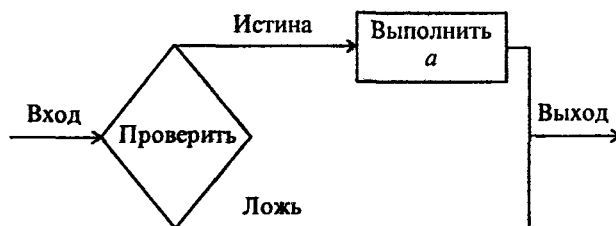


Рис. 1.21. Структура «неполное ветвление»

Цикл (или повторение) предусматривает повторное выполнение некоторого Набора команд программы. Если бы циклы не существовали, вряд ли занятие программированием было бы оправданным: циклы позволяют записать длинные последовательности операций обработки данных с помощью небольшого числа повторяющихся команд. Разновидности цикла изображены на рис. 1.22 и рис. 1.23.

Цикл начинается с проверки логического выражения. Если оно истинно, то выполняется «а», затем все повторяется снова, пока логическое выражение сохраняет значение «истина». Как только оно становится ложным, выполнение операций «а» прекращается и управление передается по программе дальше.

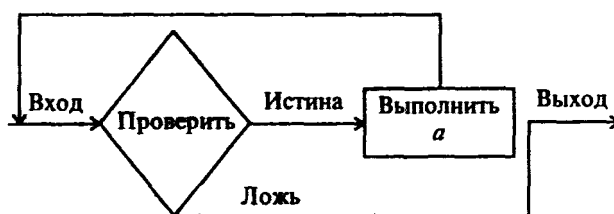


Рис. 1.22. Структура цикла «пока»

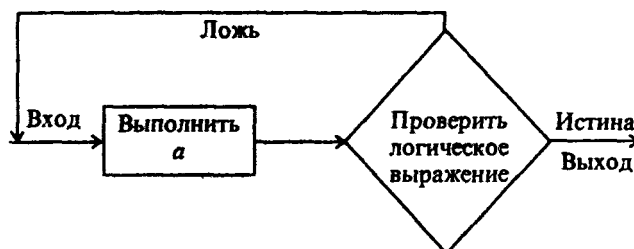


Рис. 1.23. Структура цикла «до»

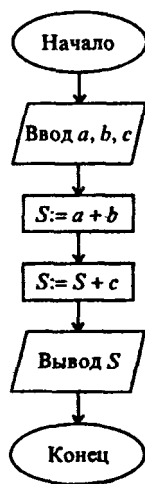


Рис. 1.24. Нахождение суммы трех чисел

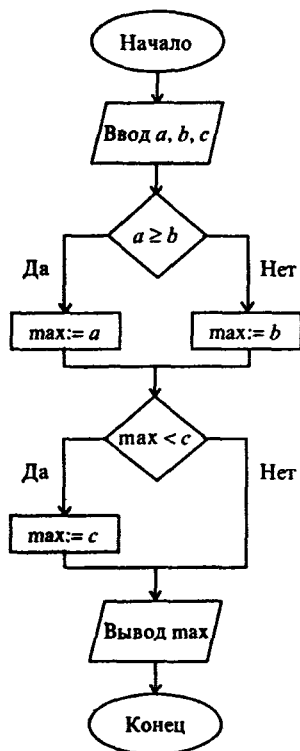


Рис. 1.25. Нахождение наибольшего из трех чисел

Эти структуры можно комбинировать одну с другой - как путем организации их следований, так и путем создания суперпозиций (вложений одной структуры в другую) - сколь угодно разнообразно для выражения логики алгоритма решения любой задачи. Используя описанные структуры, можно полностью исключить использование каких-либо еще операторов условного и безусловного перехода, что является важным признаком структурного программирования. Направление выполнения команд часто изображают сверху вниз. На рис. 1.24 - 1.26 приведены простейшие примеры структурной реализации алгоритмов работы с величинами.

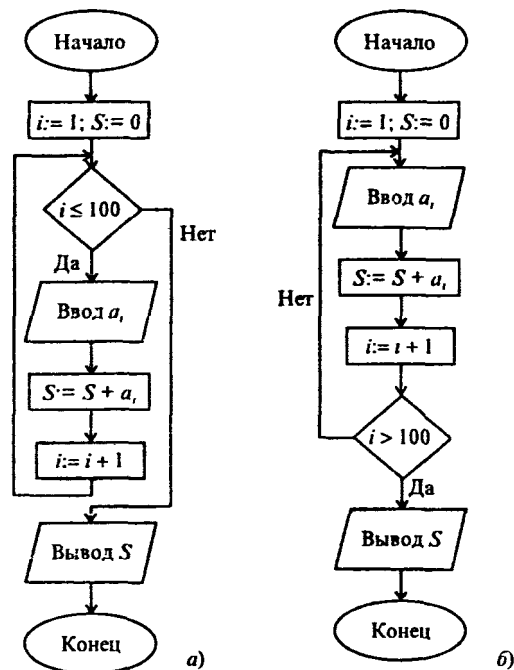


Рис. 1.26. Нахождение суммы 100 чисел

Умение образовывать из базовых структур их суперпозиции в соответствии с условиями конкретной задачи - одно из важнейших в программировании. Допустим, надо ввести в память компьютера 100 чисел и по дороге отсуммировать те из них, которые положительны. Ясно, что ввод - операция циклическая, а внутри этого цикла находится развилка, в которой проверяется знак числа и производится суммирование. Схематически соответствующая суперпозиция изображена на рис. 1.27.

Так как выражение, управляющее циклом, проверяется в самом начале, то в случае, если условие сразу окажется ложным, операторы циклической части «а» могут вообще не выполняться. Операторы циклической части «а» должны изменять переменную (или переменные), влияющие на значение логического выражения, иначе программа «заиклится» - будет выполняться бесконечно. Рассмотренная циклическая конструкция называется также цикл «пока», или «цикл с предусловием».

Существует и иная конструкция цикла, которая предусматривает проверку условия, по которому, наоборот, выполнение команд циклической части прекращается, после команд циклической части (см. рис. 1.23).

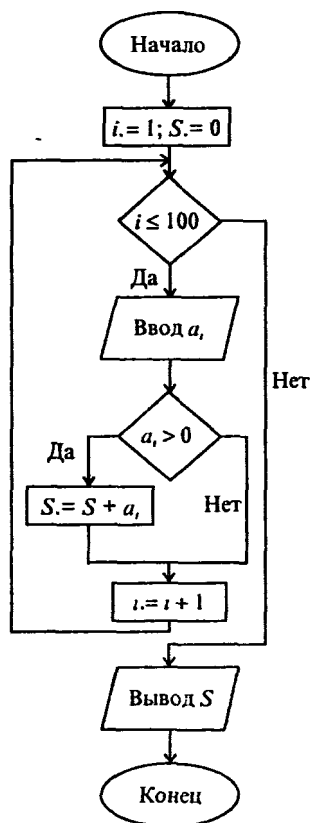


Рис 1.27 Алгоритм типа развилка, вложенная в цикл, для нахождения суммы положительных чисел из 100 возможных

Схематические изображения нескольких суперпозиций базовых алгоритмических структур представлены ниже на рис. 1.28-1.31.

Еще одним важным компонентом структурного подхода к разработке алгоритмов является модульность. Модуль - это последовательность логически связанных операций, оформленных как отдельная часть программы. Использование модулей имеет следующие преимущества:

- 1) возможность создания программы несколькими программистами;
- 2) простота проектирования и последующих модификаций программы;
- 3) упрощение отладки программы - поиска и устранения в ней ошибок;
- 4) возможность использования готовых библиотек наиболее употребительных модулей.

Но, пожалуй, самым важным достижением структурного подхода к разработке алгоритмов является нисходящее проектирование программ, основанное на идее уровней абстракции, которые становятся уровнями модулей в разрабатываемой программе. На этапе проектирования строится схема иерархии, изображающая эти уровни. Схема иерархии позволяет программисту сначала сконцентрировать внимание на определении того, что надо сделать в программе, а лишь затем решать, как это надо делать. При нисходящем проектировании исходная, подлежащая решению задача разбивается на ряд подзадач, подчиненных по своему содержанию главной задаче. Такое разбиение называется детализацией или декомпозицией.

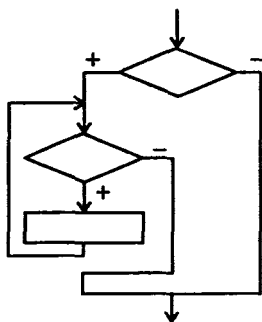


Рис. 1.28. Алгоритм типа «цикл, вложенный в неполную развилку»

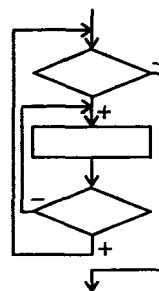


Рис. 1.29. Алгоритм типа «цикл в цикле»

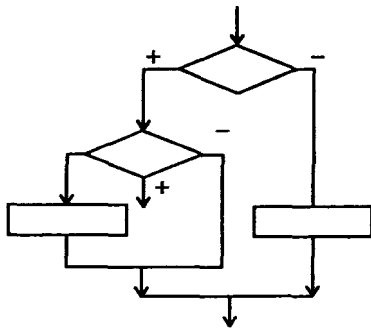


Рис. 1.30. Алгоритм типа «развилка в развилке»

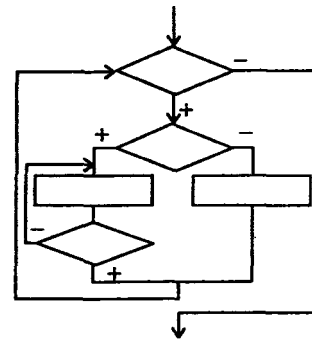


Рис. 1.31. Иллюстрация трехкратного вложения одной базовой структуры в другую

На следующем этапе эти задачи, в свою очередь, разбиваются на более мелкие подчиненные подзадачи и так далее, до уровня относительно небольших подзадач, вторые требуют для решения небольших модулей в 3 - 5 строк. Такой метод проектирования программ позволяет преодолевать проблему сложности разработки программы (и ее последующей отладки и сопровождения).

8.3. НОВЕЙШИЕ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММ ДЛЯ ЭВМ

Структурный подход сыграл огромную роль в программировании и вычислительной технике. С его использованием создан большой запас программного обеспечения, решено множество практически значимых задач. Однако, развитие программирования на этом не остановилось. Сегодня дополняющим структурное программирование, создающим основу для разработки современных аудиовизуаль-программных комплексов стало объектное (иногда говорят объектно-ориентированное) программирование, а противостоящим ему при решении определенных классов задач является декларативное программирование, выраженное двумя разными подходами - функциональным и логическим.

Здесь мы ограничимся кратчайшей характеристикой этих направлений, более детально описанных в гл. 3.

Само структурное программирование, наиболее отчетливо выраженное в языке Паскаль (PASCAL), возникло в ходе развития процедурно-ориентированного подхода, заложенного в исторически первом из языков программирования - Фортране (FORTRAN). Во всех языках этого направления разработчик алгоритма (он же, как правило, и программист) описывает, какими действиями следует реализовать процесс. В основе языков этой группы лежат понятия команд (операторов) и данных. Отдельные группы операторов могут объединяться во вспомогательные алгоритмы (процедуры, подпрограммы).

Объект - основное понятие объектного программирования - в первом приближении похож на процедуру. Однако, процедура (подпрограмма) «оживает» лишь внутри той программы, к которой она относится, а объект может вести себя вполне независимо. Он может относиться к иной предметной области нежели основная программа, быть исполненным в ином стиле. Объекты достаточно причудливо связываются друг с другом, могут перенимать свойства друг у друга («наследование»). В объектно-ориентированном подходе исходная задача представляется как совокупность взаимодействующих объектов. Наиболее популярные реализации объектного программирования созданы на основе языков Паскаль, Бейсик (BASIC).

Декларативный подход в разработке компьютерных программ появился в начале 70-х годов. Он не получил столь широкого применения как процедурный, поскольку был направлен на относительно узкий круг задач искусственного интеллекта. При его применении программист описывает свойства исходных данных, их взаимосвязи, свойства, которыми должен обладать результат, а не алгоритм получения результата. Разумеется, для получения результата этот алгоритм все равно нужен, но он должен порождаться автоматически той системой, которая поддерживает декларативно-ориентированный язык программирования. При логическом варианте такого подхода (прежде всего это относится к языку Пролог, PROLOG) задача описывается совокупностью фактов и правил в некотором формальном логическом языке, при функциональном варианте - в виде функциональных соотношений между фактами (язык Лисп, LISP).

Процедурно-ориентированное программирование развивается и в другом направлении - так

называемого, параллельного программирования. В привычных алгоритмах и программах действия совершаются последовательно одно за другим. Однако, логика решения множества задач вполне допускает одновременное выполнение нескольких операций, что ведет к многократному увеличению эффективности. Реализация параллельных алгоритмов на ЭВМ стала возможной с появлением многопроцессорных компьютеров, в которых специалисты видят будущее вычислительной техники.

Контрольные вопросы и задания

1. Какие требования предъявлялись к алгоритмам для компьютеров первых поколений?
2. Какой подход к созданию алгоритмов называется операциональным?
3. Охарактеризуйте операции, которые использовались при разработке программ при операциональном подходе.
4. Прodelайте программу вычисления $\sqrt{x}$ из текста с помощью ручки, бумаги и калькулятора.
5. В чем состоят недостатки операционального подхода к программированию?
6. Охарактеризуйте базовые структуры алгоритмов.
7. В чем состоит модульность при структурной разработке алгоритмов?
8. Что такое нисходящее проектирование программ?

§ 9. СТРУКТУРЫ ДАННЫХ

9.1. ДАННЫЕ И ИХ ОБРАБОТКА

Суть всех алгоритмов (и компьютерных программ) состоит в том, что они описывают преобразование некоторых начальных данных в конечные. Какие-то данные алгоритм (программа) может использовать как промежуточные. Естественно, что представление и организация данных имеет при разработке алгоритма первостепенное значение. Вопрос о том, как должны быть организованы данные, необходимо решать до того, как начинается разработка алгоритма (программы). Ведь прежде, чем выполнить какие-либо операции, надо иметь объекты, к которым они будут применяться, и четко представлять себе структуру объектов, которые будут получены.

Развитие вычислительной техники и программирования сопровождалось эволюцией представлений о роли данных и их организации. Одним из свойств компьютеров является способность хранить огромные объемы информации и обеспечивать легкий доступ к ней. Информация, подлежащая обработке, в некотором смысле представляет абстракцию фрагмента реального мира. Мы говорим о данных как об абстрактном представлении реальности, поскольку некоторые свойства и характеристики реальных объектов при этом игнорируются (как несущественные для данной задачи). Например, каждый сотрудник в списке сотрудников некоторого учреждения представлен множеством данных. Это множество может включать идентифицирующие данные (например, фамилию), данные, относящиеся к тому, что сотрудник делает или к тому, что делают для него. Однако маловероятно, что будут включены такие сведения, как цвет глаз или волос, рост и вес.

Решая конкретную задачу, необходимо выбрать множество данных, представляющих реальную ситуацию. Затем надлежит выбрать способ представления этой информации. Представление данных определяется исходя из средств и возможностей, допускаемых компьютером и его программным обеспечением. Однако очень важную роль играют и свойства самих данных, операции, которые должны выполняться над ними. С развитием вычислительной техники и программирования средства и возможности представления данных получили большое развитие и теперь позволяют использовать как простейшие неструктурированные данные, так и данные более сложных типов, полученные с помощью комбинации простейших данных. Такие данные называют **структурированными**, поскольку они обладают некоторой организацией. Современные средства программирования позволяют оперировать с множествами, массивами, записями, файлами (очередями).

В более сложных случаях программист может задать динамические структуры данных, память для хранения которых выделяется прямо в процессе выполнения программы. К таким данным относят линейные списки (одно- и двунаправленные), стеки, деревья, графы.

В последние годы получило развитие, так называемое, объектно-ориентированное программирование, в котором в известной мере устранено противостояние . данных и программ. Объект - это некое образование, состоящее не только из данных, но и из процедур их обработки.

Остановимся подробнее на свойствах различных представлений данных или, как | еще говорят, типах данных.

9.2. ПРОСТЫЕ (НЕСТРУКТУРИРОВАННЫЕ) ТИПЫ ДАННЫХ

В математике принято классифицировать величины в соответствии с их характеристиками. Различают целые, вещественные, комплексные и логические величины, величины, представляющие собой отдельные значения, множества значений или множества множеств. Аналогично этому в информатике любая константа, переменная, выражение или функция относится к некоторому типу. Фактически тип характеризует множество значений, которые может принимать константа, переменная, выражение или функция. Широко используется правило, по которому тип явно указывается в описании константы, переменной или функции. К данным каждого типа применимы определенные операции и их поведение подчиняется некоторым аксиомам.

Так, над целыми числами могут выполняться операции сложения (+), вычитания (-) и умножения (*). Существуют две различные операции, связанные с делением и не выводящие за границы множества целых чисел: 1) определение целой части от деления одного числа на другое; 2) определение остатка от деления одного числа на другое.

Целые числа, используемые компьютером, имеют те же свойства (подчиняются тем же аксиомам), что и целые числа в арифметике. Все вычисления с ними выполняются абсолютно точно (не приближенно). Имеется только одно отличие в свойствах компьютерных целых чисел от тех, которыми оперирует абстрактная математика, а именно ограниченность диапазона: для каждой компьютерной системы имеется самое большое допустимое в ней целое число $M_{+\infty}$ (и самое малое, отрицательное $M_{-\infty}$). Обычно выполняется соотношение

$$M_{+\infty} + 1 = M_{-\infty} \quad (M_{-\infty} - 1 = M_{+\infty}),$$

т. е. прибавив единицу к самому большому допустимому положительному числу, мы получим модуль самого малого допустимого отрицательного. Это свойство компьютерных чисел связано с особенностями их кодирования в ячейках памяти компьютера.

Над действительными (или вещественными) числами могут быть выполнены операции сложения (+), вычитания (-), умножения (*) и деления (/), так же, как и над математическими действительными числами. Однако, все операции над действительными числами выполняются с точностью, не превосходящей некоторого фиксированного значения, вследствие того, что представления чисел в памяти компьютера имеют ограниченную длину (зависящую от конкретного компьютера и используемой системы программирования). Так, например, в машинной арифметике может быть $1/3 = 0,33333333$, тогда как математически точное десятичное представление дроби $1/3$ - бесконечно длинное число.

Главным свойством литерных (символьных) данных является их упорядоченность, т.е. свойство быть сравнимыми. Обычным признаком значения символьной или текстовой величины являются кавычки, и справедливо ' a ' < ' b ', ' b ' < ' c ', ' c ' < ' d ' и т.д. Каждый символ имеет определенный числовой код (например, код символа латинской буквы "А" в большинстве кодировок 63) и упорядочение происходит в соответствии с этими числовыми кодами. Как правило, имеются функции, позволяющие получить по символу его код и символ по коду.

Для строчных величин единственной выполнимой операцией является конкатенация («сложение») строк. Например, ' $abcd$ ' + ' efg ' = ' $abcdefg$ '. В конкретных системах обычно определены функции, определяющие длину строк, вхождение в них тех или иных подстрок, вырезающие из строк некоторые фрагменты.

К логическим данным, способным принимать значения «истина» («true») или «ложь» («false»), применимы основные операции логики высказываний: конъюнкция (логическое «и» - «and»), дизъюнкция (логическое «или» - «or»), отрицание (логическое «не» - «not»). Иногда можно использовать операции импликации («если»), эквиваленции («если и только если») и т.п. Эти логические операции определяются таблицей истинности, табл. 1.9.

Таблица 1.9 Таблица истинности для логических операций

A	B	«и» (A and B)	«или» (A or B)	«не» (not A)
И	И	И	И	Л
И	Л	Л	И	Л
Л	И	Л	И	И
Л	Л	Л	Л	И

Часто при решении задач оказываются полезными определенные разработчиком типы данных, представляющие собой перечисление некоторых констант или ограниченный с обеих сторон диапазон определенных ранее данных. Например, создается тип данных из двух символов 'a' и 'b' или из целых чисел в диапазоне от 1 до 100 и т. д.

9.3. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ

Описанные выше типы данных называют простыми. Основной признак, по которому можно определить величину простого типа, таков: одно имя - одно значение.

Значительно большие возможности заключают в себе структурированные данные, определяемые разработчиком программы (в пределах возможностей используемого им языка программирования). К структурированию данных разработчика программы толкает как логика прикладной задачи, так и чисто утилитарное соображение: при наличии в задаче большого количества входных и выходных данных отдельное именование каждого из них может оказаться практически невозможным.

Разумеется, действия разработчика алгоритма и программы ограничены возможностями того языка программирования, на который он ориентируется. В разных языках возможности структуризации переменных на уровне сложных структур не совпадают, но многие структуры давно стали традиционными и реализованы в большинстве практически используемых языков программирования.

Структурированные типы данных классифицируют по следующим основным признакам: однородная - неоднородная, упорядоченная - неупорядоченная, прямой доступ - последовательный доступ, статическая - динамическая. Эти признаки противостоят друг другу лишь внутри пары, а вне этого могут сочетаться.

Если все элементы, образующие структуру, однотипны (например - целые числа или символы), то структура является однородной; если же в ней «перепутаны» элементы разной природы (например, числа чередуются с символами), то неоднородной.

Структуру называют **упорядоченной**, если, между ее элементами определен порядок следования. Примером упорядоченной математической структуры служит числовая последовательность, в которой у каждого элемента (кроме первого) есть предыдущий и последующий. Наличие индекса в записи элементов структуры уже указывает на ее упорядоченность (хотя индекс для этого не является обязательным признаком).

По способу доступа упорядоченные структуры бывают прямого и последовательного доступа. При прямом доступе каждый элемент структуры доступен пользователю в любой момент независимо от других элементов. Глядя на линейную таблицу чисел мы можем списать или заменить сразу, допустим, десятый элемент. Однако, если эта таблица не на бумаге, а, скажем, каким-то образом записана на магнитофонную ленту, то сразу десятое число нам недоступно - надо сначала извлечь девять предшествующих. В последнем случае мы имеем дело с последовательным доступом.

Если у структуры размер (длина, количество элементов) не может быть изменен «на ходу», а фиксирован заранее, то такую структуру называют статической. Программные средства информатики иногда позволяют не фиксировать размер структуры, а устанавливать его по ходу решения задачи и менять при необходимости, что бывает очень удобно. Такую структуру называют динамической. Например, при описании закономерностей движения очереди в магазине мы не знаем заранее, сколько человек в ней будет в тот или иной момент, и соответствующую структуру дан-

ных (например, список фамилий участников очереди) лучше представлять динамической.

Массивы

Самым традиционным и широко известным из структурированных типов данных является **массив** (иначе называемый регулярным типом) - однородная упорядоченная статическая структура прямого доступа.

Массивом называют однородный набор величин одного и того же типа, называемых компонентами массива, объединенных одним общим именем (**идентификатором**) и идентифицируемых (адресуемых) вычисляемым **индексом**. Это определение подчеркивает, что все однотипные компоненты массива имеют одно и то же имя, но различаются по индексам, которые могут иметь характер целых чисел из некоторого диапазона, литер, перечисленных констант. Индексы позволяют адресовать компоненты массива, т.е. получить доступ в произвольный момент времени к любой из них как к одиночной переменной (рис. 1.32). Обычный прием работы с массивом - выборочное изменение отдельных его компонент.

Вычисляемые индексы позволяют использовать единое обозначение элементов массива для описания массовых однотипных операций в циклических конструкциях программ. Важной особенностью массива является его статичность. Массив должен быть описан в программе (т.е. определены тип и число компонент) и его характеристики не могут быть изменены в ходе выполнения программы.

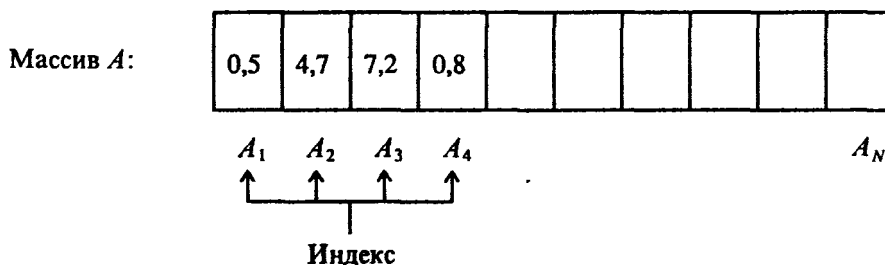


Рис. 1.32. Одномерный массив - набор элементов (компонентов)

Компонентами массива могут быть не только простейшие данные, но и структурные, в том числе массивы. В этом случае мы получаем массив массивов - многомерный **массив**. Для индексации элементарных компонент в этом случае может потребоваться два, три и более индексов.

В некоторых системах программирования существуют специальные виды массивов. Например, массив литер (символов) определяется как строка.

Данные, хранящиеся в массивах, находятся в оперативной памяти компьютера. Это, с одной стороны, ускоряет доступ к ним в ходе решения задачи, а с другой - налагает ограничения на объем возможной информации, организованной в виде массивов. Не следует поэтому, без крайней необходимости, создавать новые массивы для перемещения данных из уже существующих массивов.

Рассмотрим в качестве примера задачу сортировки набора некоторых данных, для которых имеют смысл отношения «больше» или «меньше». Представьте себе, что надо карточки в картотеке разместить в порядке возрастания записанных на них чисел. Используем для сортировки набора чисел (т.е. записи их в порядке возрастания) одномерный (линейный) массив. Дадим ему имя A , тогда $a_1, a_2, a_3, \dots, a_n$ - компоненты массива.

Существует огромное число методов сортировки массивов. Рассмотрим один из самых простых (но не самых быстрых) - метод выбора.

В начале процесса имеем заполненный числами массив (неотсортированный). Процесс сортировки строится по индукции. Допустим, мы уже отсортировали часть массива и имеем упорядоченную последовательность

$$a_1 < a_2 < \dots < a_{i-1}$$

и оставшуюся неотсортированной последовательность

$$a_i, a_{i+1}, \dots, a_N.$$

При каждом шаге, начиная с $i = 1$, из неотсортированной части последовательности извлекается наименьший элемент $x = a_i$, и меняется местами с i -м элементом. Затем этот процесс повторяется для $i = 2, i = 3$ и т.д., до тех пор пока не останется один, самый большой элемент.

Этот алгоритм потребует многократного нахождения наименьшего элемента массива. Этот «вспомогательный» алгоритм поиска наименьшего среди $a_i, \dots, a_N$ может быть следующим:

1) фиксируется в качестве значения вспомогательной переменной m первый слева элемент массива: $m = a_i$ (в конце процесса m будет иметь значение наименьшего элемента);

2) выполняется сравнение m с элементом массива a_j , (начиная с номера $j = i + 1$) и, если $a_j < m$, то m заменяется на a_j ;

3) далее выполняется сравнение m с очередным элементом массива, т.е. j увеличивается на единицу и шаги 2, 3 выполняются снова, до тех пор пока j не достигнет максимального значения индекса элемента массива.

После выполнения этих предписаний переменная m будет соответствовать наименьшему элементу массива.

Двумерный массив визуально представляется плоской таблицей, табл. 1.10. При наличии одного имени (идентификатора) для всех компонентов каждый из них фиксируется значениями двух индексов, указывающих номер строки и номер столбца, на пересечении которых находится эта компонента.

Рассмотрим пример обработки данных, хранящихся в двумерном массиве. Допустим, что на некоторой территории (например, страны) «квадратно-гнездовым» способом расставлены температурные датчики, и их показания обраны в одном центре (что вполне близко к реальной деятельности метеослужбы). Тогда в таблицу - двумерный массив - попадут значения температуры t_{ij} в соответствующих точках. Требуется, просматривая таблицу построчно, найти те точки (т.е. индексы узлов), между которыми температура принимает некоторое заданное значение T .

Таблица 1.10 Графический образ двумерного массива

$i \backslash j$	1	2	3	4	...
1	a_{11}	a_{12}	a_{13}	a_{14}	...
2	a_{21}	a_{22}	a_{23}	a_{24}	...
3	a_{31}	a_{32}	a_{33}	a_{34}	...
4	a_{41}	a_{42}	a_{43}	a_{44}	...
...	...	...	...	...	...

Пусть в таблице n строк и m столбцов. Вспомогательным алгоритмом в данной задаче может быть алгоритм поиска нужных узлов в одной строке. Пусть эта строка имеет номер k . Алгоритмы записаны без комментариев для самостоятельного разбора.

Вспомогательный алгоритм (k):

- 1) положить $j = 1$;
- 2) если $t_{kj} < T < t_{kj+1}$, то см. п. 2;
- 3) увеличить j на 1,
- 4) если $j < m$, то вернуться к п. 2;
- 5) задача решена, ответ: $(k, j), (k, j + 1)$;
- 6)конец.

Основной алгоритм:

- 1) положить $k = 1$;
- 2) выполнить вспомогательный алгоритм (K);
- 3) увеличить k на 1;
- 4) если $k > n$, то вернуться к п.2;
- 5)конец.

Записи, множества, файлы

Обобщением массива является комбинированный тип данных - **запись**, являющаяся неоднородной упорядоченной статической структурой прямого доступа. Запись есть набор именованных компонент - **полей** (часто разного типа), объединенных одним общим именем и идентифицируемых (адресуемых) с помощью как имени записи, так и имен полей, рис. 1.33.

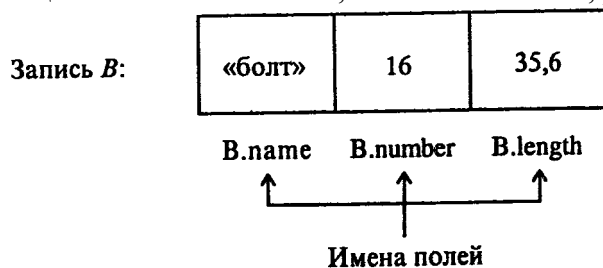


Рис. 1.33. Иллюстрация «записи».

Запись *B* состоит из трех полей, имеющих последовательно типы «текст», «целое число», «вещественное число»: 1-е поле - название детали, 2-е - условный номер по каталогу, 3-е - длина. При работе с одной единственной записью (что бывает нечасто), имя поля можно использовать как обычную переменную, т.е. можно изменять значение поля с помощью операции присваивания или любых других операций, доступных над величинами данного типа. Если же данная запись - лишь часть набора данных, то имя поля состоит из двух частей и называется **составным именем поля** (на рис. 1.33 составные имена *B.name*, *B.number*, *B.length*).

Для облегчения работы с полями в различных языках программирования существуют средства, облегчающие их адресацию.

И записи, и массивы обладают одним общим свойством - произвольным доступом к компонентам. Записи более универсальны в том смысле, что для них не требуется идентичности типов их компонент. Массивы обеспечивают большую гибкость - индексы их компонент можно вычислять в отличие от имен полей записей.

Существенно иные возможности дает структура данных, моделирующая свойства математического объекта - множества.

Над множеством могут быть выполнены следующие операции:

- 1) объединение множеств (операция сложения '+');
- 2) пересечение множеств (операция умножения '*');
- 3) теоретико-множественная разность (вычитание множеств '-');
- 4) проверка принадлежности элемента множеству.

Различия между множеством и массивом очень существенны: размер множества заранее не оговаривается (хотя и ограничен компьютерной реализацией, например, 255), не существует иного способа доступа к элементам множества, кроме как проверкой принадлежности множеству.

Более сложной, чем рассмотренные выше из предусмотренных в современных системах программирования структур данных, является очередь (**файл**).

Понятие «файл» при всей своей привычности употребляется в информатике в нескольких не совсем совпадающих смыслах. Здесь мы остановимся лишь на представлении о файле как однородной упорядоченной динамической структуре последовательного доступа - очереди.

Очередь есть линейно упорядоченный набор следующих друг за другом компонент, доступ к которым происходит по следующим правилам:

- 1) новые компоненты могут добавляться лишь в хвост очереди;
- 2) значения компонент могут читаться (извлекаться) лишь в порядке следования компонент от головы к хвосту очереди.

Размер очереди заранее не оговаривается и теоретически может считаться бесконечным. Для запоминания (хранения) компонент очереди часто используют внешние запоминающие устройства большой емкости - магнитные диски и ленты. Отсюда другое название очереди - файл (по английски это слово имеет ряд значений, в том числе «картотека», «шеренга», «очередь»).

Исторически слово «файл» стало впервые применяться в информатике для обозначения по-

следовательного набора каких-либо данных или команд (программа), хранящихся на внешнем запоминающем устройстве. Несколько позже были осознаны абстрактные, не зависящие от магнитных дисков и лент, свойства очереди как структуры данных, полезные при решении многих задач обработки - информации. Такой принцип извлечения и добавления компонент к очереди часто; называется «первым вошел - первым вышел» (английская аббревиатура - «FIFO»), рис. 1.34.

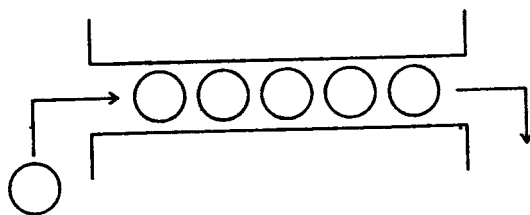


Рис. 1.34. Иллюстрация «очереди»

В языках программирования существуют и такие разновидности файлов, которые не подчиняются условию последовательности доступа к его компонентам (так называемые, файлы прямого доступа). Они уже не являются очередями.

Суперпозиция структур данных

Из рассмотренных структур данных можно создавать различные суперпозиции (вопрос о допустимости той или иной суперпозиции в конкретном языке программирования следует искать в его описании).

Рассмотрим в качестве примера такую часто используемую суперпозицию как **файл записей** - обычную, например, при создании баз данных. Итак, имеется файл по имени F, содержащий некоторое количество таких записей, как на рис. 1.30. Составим алгоритм подсчета количества болтов, у которых длина (length) заключена в пределах от 3 до 40:

- 1) положить $k = 0$ (в конце работы k - число искомых болтов);
- 2) прочесть первую запись из файла;
- 3) если $B.name = 'болт'$ и $30 < B.length < 40$, то увеличить k на 1;
- 4) если файл уже опустел, то идти к п. 7, иначе - к п. 5;
- 5) прочесть следующую запись из файла;
- 6) идти к п.3;
- 7) конец работы; k - число искомых болтов.

Стек

Существует (и часто используется) и другая структура данных, в которой тот элемент, который первый в нее помещался, выходит последним и, наоборот, тот, который последним входит, выходит первым (английская аббревиатура «LIFO»). Такая структура получила название **стек** (или магазин - по сходству с магазином стрелкового оружия), рис. 1.35.

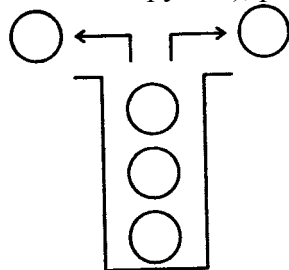


Рис. 1 35. Иллюстрация «стека»

Стеки и принцип LIFO находят очень широкое применение в информатике. Рассмотрим в качестве примера использование стека при вычислении значения арифметического выражения.

Вычисление значения выражения требует соблюдения старшинства операций. Операции по старшинству при вычислении значений математических выражений располагаются в следующем порядке: вычисление значений функций (включая возведения в степень), умножения и деления, сложения и вычитания. Изменить такой «естественный» порядок операций можно с помощью

скобок.

Например, вычисление известного из школьного курса математики выражения $b^2 - 4*a*c$ включает предварительное установление порядка выполнения операций:

$$\begin{array}{ccc} 14 & 2 & 3 \\ b^2 & - & 4*a*c \end{array}$$

Для этого выражение просматривают несколько раз. Выполнение каждой операции дает некоторое число, которое приходится записывать отдельно от выражения, запоминая тот фрагмент выражения, для которого число является значением.

Сейчас рассмотрим экономный алгоритм вычисления значения выражения, использующий два магазина для перестановки элементов выражения (с учетом старшинства операций) и для хранения промежуточных результатов. Магазины обозначим M_1 и M_2 , в M_1 будут попадать знаки операций, в M_2 - числа, участвующие в записи выражения, значения переменных и все промежуточные числовые значения.

Ограничимся выражениями, состоящими только из чисел и переменных без индекса, связанных знаками операций, *, /, +, -. Знак «минус» будет знаком лишь двухместной операции вычитания, выражения типа « - $a + 1$ » исключаются из рассмотрения. От этих ограничений можно было бы и отказаться, но это удлинит изложение. Пока предположим также, что в выражении нет скобок.

Опишем алгоритм вычисления. Исходное выражение читается слева направо; если прочитано число, то оно заносится в M_2 , если переменная - в M_2 заносится ее значение; если же прочитан знак операции, то необходимо различать несколько случаев.

- 1) M_1 пуст; прочитанный знак помещается на вершину M_1 .
- 2) прочитанный знак помещается на вершину M_1 , если он обозначает операцию, которая старше и поэтому должна выполняться до операции, знак которой был расположен на вершине M_1 .
- 3) если операции равноправны или если та, знак которой только что прочитан в выражении, должна выполняться позднее, необходимо применить операцию, знак которой расположен на вершине M_1 , к двум верхним числам из M_2 (число на вершине - второй операнд, число под ним - первый); знак операции на вершине M_1 удаляется из M_1 , вместо двух верхних чисел в M_2 помещается результат выполнения над ними операции.

В некоторый момент в исходном выражении не остается символов. Если пуст и M_1 , то вычисление окончено, результат находится в M_2 ; в противном случае знаки операции извлекаются по очереди из M_1 и соответствующие операции применяются к числам из M_2 .

Рассмотрим вычисление выражения $b^2 - 4*a*c$; значения переменных a, b, c обозначим A, B, C . Знак возведения в степень обозначим, как часто делается, стрелкой вверх.

$$\begin{array}{ll} M_2: & B \\ M_2: & 2 \\ & B \end{array} \qquad \begin{array}{ll} M_1: & \uparrow \\ M_1: & \uparrow \end{array}$$

$M_2:$	$B \uparrow 2$	$M_1:$	
$M_2:$	4 $B \uparrow 2$	$M_1:$	$-$
$M_2:$	A 4 $B \uparrow 2$	$M_1:$	$*$ $-$
$M_2:$	$4 * A$ $B \uparrow 2$	$M_1:$	$-$
$M_2:$	C $4 * A$ $B \uparrow 2$	$M_1:$	$*$ $-$
$M_2:$	$4 * A * C$ $B \uparrow 2$	$M_1:$	$-$
$M_2:$	$B \uparrow 2 - 4 * A * C$	$M_1:$	

Про знак операции говорят, что он имеет более высокий приоритет в сравнении с другим знаком, если обозначаемая им операция старше. В других случаях говорят о равных приоритетах или более низком приоритете. Рассмотренные знаки операций распадаются на группы равных по приоритету:

$\uparrow$
 $*, /$
 $+, -$

Группы упорядочены по убыванию приоритета.

Теперь дадим правило работы со скобками. Левая скобка заносится в M_1 сразу после прочтения. Прочтение правой скобки влечет выполнение всех операций, знаки которых находятся в M_i выше левой скобки; после выполнения этих операций обе скобки уничтожаются. Вот что будет происходить при выполнении $(a + b) * c$:

$M_2:$	A	$M_1:$	$($
$M_2:$	B A	$M_1:$	$+$ $($
$M_2:$	$A + B$	$M_1:$	$($
$M_2:$	$A + B$	$M_1:$	
$M_2:$	C $A + B$	$M_1:$	$*$
$M_2:$	$(A + B) * C$	$M_1:$	

Иерархическая организация данных

Во всех рассмотренных выше структурах отдельные элементы (компоненты, поля, составляющие) структуры были формально равноправны. Существует, однако, широкий круг задач, в которых одни данные естественным образом «подвязаны» к другим. В этом случае возникает соподчиненная (иерархическая) структура данных. Ограничимся конкретным примером. Представим себе генеалогическое дерево, корень которого - имя человека, на следующем уровне - имена его родителей, еще на следующем - имена родителей родителей и т.д. Такая структура называется **двоичным деревом**, рис. 1.36.

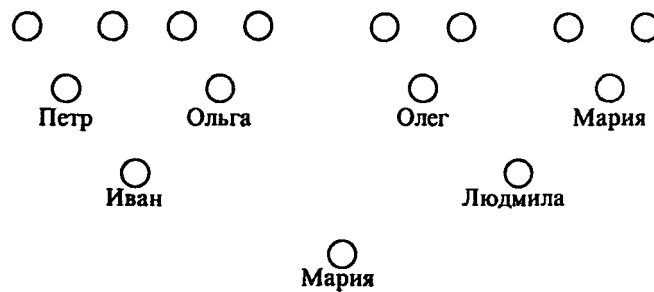


Рис. 1.36. Структура типа «двоичное дерево»;
пара ближайших по горизонтали кружков - мужское и женское имя

Как структурировать эти данные (имена)? Для помещения их в текстовый массив и запись трудно придумать логически оправданный порядок следования. Самое разумное - создать динамическую структуру типа той, что изображена на рис. 1.36. современные языки программирования позволяют это делать и обрабатывать такие структуры с высокой эффективностью.

Контрольные вопросы и задания

1. Какое значение имеет выбор представления и организации данных при разработке программы?
2. Какие данные можно отнести к простейшим неструктурированным?
3. Какие данные называют структурированными?
4. Охарактеризуйте свойства данных целого, действительного типа.
5. Какими свойствами обладают литерные и строковые величины?
6. Что называют логическими данными?
7. Что такое массив?
8. Решение каких задач требует использования массивов?
9. Разберите алгоритм сортировки массива, приведенный в тексте. Предложите свой алгоритм, отличный от данного.
10. Как определяется тип «множество»?
11. Чем отличается комбинированный тип данных (запись) от массива?
12. Что такое очередь (файл)? Какое применение имеют файлы?
13. Что такое стек?
14. Разберите по аналогии с примером, имеющимся в тексте о стеке, вычисление других арифметических выражений с использованием стеков.
15. Придумайте примеры естественной иерархической организации данных.

§ 10. ПОНЯТИЕ ОБ ИНФОРМАЦИОННОМ МОДЕЛИРОВАНИИ

10.1. МОДЕЛИРОВАНИЕ КАК МЕТОД РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

С точки зрения информатики, решение любой производственной или научной задачи описывается следующей технологической цепочкой: «реальный объект - модель - алгоритм - программа - результаты - реальный объект». В этой цепочке очень важную роль играет звено «модель», как необходимый, обязательный этап решения этой задачи. Под моделью при этом понимается некоторый мысленный образ реального объекта (системы), отражающий существенные свойства объекта и заменяющий его в процессе решения задачи.

Модель - очень широкое понятие, включающее в себя множество способов представления изучаемой реальности. Различают модели материальные (натурные) и идеальные (абстрактные). Материальные модели основываются на чем-то объективном, существующем независимо от человеческого сознания (каких-либо телах или процессах). Материальные модели делят на физические (например авто- и авиамодели) и аналоговые, основанные на процессах, аналогичных в каком-то отношении изучаемому (например, процессы в электрических цепях оказываются аналогичными многим механическим, химическим, биологическим и даже социальным процессам и могут быть использованы для их моделирования). Границу между физическими и аналоговыми моделями

провести можно весьма приблизительно и такая классификация моделей носит условный характер.

Еще более сложную картину представляют идеальные модели, неразрывным образом связанные с человеческим мышлением, воображением, восприятием. Среди идеальных моделей можно выделить интуитивные модели, к которым относятся, например, произведения искусства - живопись, скульптура, литература, театр и т.д., но единого подхода к классификации остальных видов идеальных моделей нет. Иногда эти модели все разом относят к информационным. В основе такого подхода лежит расширительное толкование понятия «информация»: «информацией является почти все на свете, а может быть, даже вообще все». Такой подход является не вполне оправданным, так как он переносит информационную природу познания на суть используемых в процессе моделей - при этом любая модель является информационной. Более продуктивным представляется такой подход к классификации идеальных моделей, при котором различают следующие.

1. Вербальные (текстовые) модели. Эти модели используют последовательности предложений на формализованных диалектах естественного языка для описания той или иной области действительности (примерами такого рода моделей являются милицкий протокол, правила дорожного движения, настоящий учебник).

2. Математические модели - очень широкий класс знаковых моделей (основанных на формальных языках над конечными алфавитами), широко использующих те или иные математические методы. Например, можно рассмотреть математическую модель звезды. Эта модель будет представлять собой сложную систему уравнений, описывающих физические процессы, происходящие в недрах звезды. Математической моделью другого рода являются, например, математические соотношения, позволяющие рассчитать оптимальный (наилучший с экономической точки зрения) план работы какого-либо предприятия.

3. Информационные модели - класс знаковых моделей, описывающих информационные процессы (возникновение, передачу, преобразование и использование информации) в системах самой разнообразной природы.

Граница между вербальными, математическими и информационными моделями может быть проведена весьма условно; возможно, информационные модели следовало бы считать подклассом математических моделей. Однако, в рамках информатики как самостоятельной науки, отдельной от математики, физики, лингвистики и других наук, выделение класса информационных моделей является целесообразным. Информатика имеет самое непосредственное отношение и к математическим моделям, поскольку они являются основой применения компьютера при решении задач различной природы: математическая модель исследуемого процесса или явления на определенной стадии исследования преобразуется в компьютерную (вычислительную) модель, которая затем превращается в алгоритм и компьютерную программу, рис. 1.37.



Рис. 1.37. Обобщенная схема компьютерного математического моделирования

10.2. ОСНОВНЫЕ ПОНЯТИЯ ИНФОРМАЦИОННОГО МОДЕЛИРОВАНИЯ

Остановимся на информационных моделях, отражающих процессы возникновения, передачи, преобразования и использования информации в системах различной природы. Начнем с определения простейших понятий информационного моделирования.

Экземпляр будем называть представление предмета реального мира с помощью некоторого набора его характеристик, существенных для решения данной информационной задачи (служашей контекстом построения информационной модели). Множество экземпляров, имеющих одни и те же характеристики и подчиняющиеся одним и тем же правилам, называется объектом.



Рис. 1.38. Пример абстрагирования при построении информационной модели

Таким образом, объект есть абстракция предметов реального мира, объединяемых общими характеристиками и поведением, рис. 1.38.

Информационная модель какой-либо реальной системы состоит из объектов. Каждый объект в модели должен быть обеспечен уникальным и значимым **именем** (а также идентификатором, служащим ключом для указания этого объекта, связи его с другими объектами модели). Таким образом обозначение, наименование объекта - это элементарная процедура, лежащая в основе информационного моделирования.

Объект представляет собой один типичный (но неопределенный) экземпляр чего-то в реальном мире и является простейшей информационной моделью. Объекты представляют некие «сущности» предметов реального мира, связанные с решаемой задачей.

Большинство объектов, с которыми приходится встречаться, относятся к одной из следующих категорий:

- реальные объекты;
- роли;
- события;
- взаимодействия;
- спецификации.

Реальный объект - это абстракция физически существующих предметов. Например, на автомобильном заводе это кузов автомобиля, двигатель, коробка передач; при перевозке грузов это контейнер, средство перевозки.

Роль - абстракция цели или назначения человека, части оборудования или учреждения (организации). Например, в университете как в учебном заведении это студент, преподаватель, декан; в университете как в учреждении это приемная комиссия, отдел кадров, бухгалтерия, деканат.

Событие - абстракция чего-то случившегося. Например, поступление заявления от абитуриента в приемную комиссию Университета, сдача (или несдача) экзамена.

Взаимодействия - объекты, получаемые из отношений между другими объектами. Например, сделка, контракт (договор) между двумя сторонами, свидетельство об образовании, выдаваемое учебным заведением его выпускнику.

Объекты-спецификации используются для представления правил, стандартов или критериев качества. Например, перечень знаний, умений и навыков выпускника математического факультета, рецепт проявления фотопленки.

Для каждого объекта должно существовать его **описание** - короткое информационное утверждение, позволяющее установить, является некоторый предмет экземпляром объекта или нет. Например, описание объекта «Абитуриент университета» может быть следующим: человек в возрасте до 35 лет, имеющий среднее образование, подавший в приемную комиссию документы и заявление о приеме.

Предметы реального мира имеют **характеристики** (такие, например, как имя, название, регистрационный номер, дата изготовления, вес и т.д.). Каждая отдельная характеристика, общая для всех возможных экземпляров объекта, называется атрибутом. Для каждого экземпляра атрибут принимает определенное значение. Так, объект **Книга** имеет атрибуты **Автор, Название, Год издания, Число страниц**.

У каждого объекта должен быть **идентификатор** - множество из одного или более атрибутов, значения которых определяют каждый экземпляр объекта. Для книги атрибуты Автор и Название совместно образуют идентификатор. В тоже время Год издания и Число страниц идентификаторами быть не могут - ни врозь, ни совместно, так как не определяют объект. Объект мо-

жет иметь и несколько идентификаторов, каждый из которых составлен из одного или нескольких атрибутов. Один из них может быть выбран как привилегированный для соответствующей ситуации.

Объект может быть представлен вместе со своими атрибутами несколькими различными способами. Графически объект может быть изображен в виде рамки, содержащей имя объекта и имена атрибутов. Атрибуты, которые составляют привилегированный идентификатор объекта, могут быть выделены (например, символом * слева от имени атрибута):

Книга	
* Автор	
* Название	
. Год издания	
. Количество страниц	

В эквивалентном текстовом представлении это может иметь следующий вид:

Книга (Автор. Название. Год издания. Число страниц).

Привилегированный идентификатор подчеркивается.

Еще одним способом представления объекта информационной модели является таблица. В этой интерпретации каждый экземпляр объекта является строкой в таблице, а значения атрибутов, соответствующих каждому экземпляру, - клетками строки, табл. 1.11.

Таблица 1.11 Таблица как представление информационной модели

Автор	Книга		
	Название	Год издания	Число страниц
Грин А.	Бегущая по волнам	1988	279
Стивенсон Р. П.	Остров сокровищ	1992	269
Скотт В.	Ричард Львиное Сердце	1993	349
Гончаров И. А.	Обрыв	1986	598

Можно классифицировать атрибуты по принадлежности к одному из трех различных типов:

- описательные;
- указывающие;
- вспомогательные.

Описательные атрибуты представляют факты, внутренне присущие каждому экземпляру объекта. Если значение описательного атрибута изменится, то это говорит о том, что некоторая характеристика экземпляра изменилась, но сам экземпляр остался прежним.

Указательные атрибуты могут использоваться как идентификаторы (или часть идентификаторов) экземпляра. Если значение указывающих атрибутов изменяется, то это говорит лишь о том, что новое имя дается тому же самому экземпляру.

Вспомогательные атрибуты используются для связи экземпляра одного объекта с экземпляром другого объекта.

Рассмотрим пример:

Автомобиль
* гос. номер
. марка
. цвет
. владелец

Атрибут «цвет» является описательным, атрибуты «гос. номер» и «марка» - указательными, атрибут «владелец» - вспомогательным, служащим для связи экземпляра объекта Автомобиль с экземпляром объекта Автолюбитель. Если значение вспомогательного атрибута изменится, это говорит о том, что теперь другие экземпляры объектов связаны между собой.

10.3. СВЯЗИ МЕЖДУ ОБЪЕКТАМИ

В реальном мире между предметами существуют различные отношения. Если предметы

моделируются как объекты, то отношения, которые систематически возникают между различными видами объектов, отражаются в информационных моделях как связи. Каждая **связь** задается в модели определенным именем. Связь в графической форме представляется как линия между связанными объектами и обозначается идентификатором связи.

Существует три вида связи: один-к-одному (рис. 1.39), один-ко-многим (рис. 1.40) и многие-ко-многим (рис. 1.41).

Связь один-к-одному существует, когда один экземпляр одного объекта связан с единственным экземпляром другого. Связь один-к-одному обозначается стрелками $\leftarrow$ и $\rightarrow$.

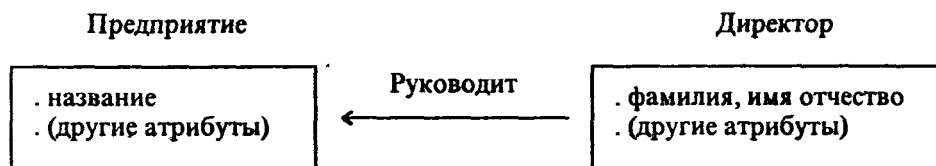


Рис. 1.39. Пример связи «один-к-одному»

Связь один-ко-многим существует, когда один экземпляр первого объекта связан с одним (или более) экземпляром второго объекта, но каждый экземпляр второго объекта связан только с одним экземпляром первого. Множественность связи изображается двойной стрелкой $\rightarrow\rightarrow$.

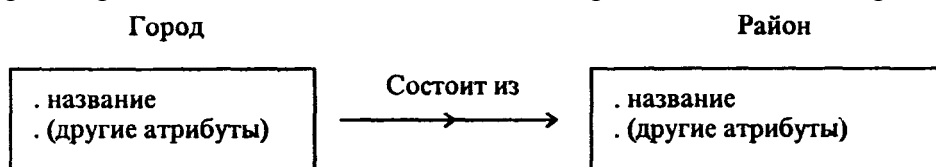


Рис. 1.40. Пример связи «один-ко-многим»

Связь многие-ко-многим существует, когда один экземпляр первого объекта связан с одним или большим количеством экземпляров второго и каждый экземпляр второго связан с одним или многими экземплярами первого. Этот тип связи изображается двусторонней стрелкой $\leftrightarrow$.

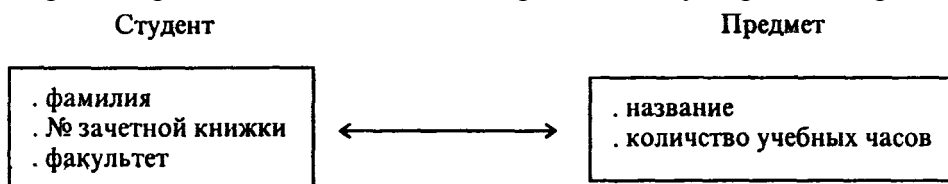


Рис. 1.41. Пример связи «многие-ко-многим»

Помимо множественности, связи могут подразделяться на безусловные и условные. В безусловной связи для участия в ней требуется каждый экземпляр объекта. В условной связи принимают участие не все экземпляры объекта. Связь может быть условной как с одной, так и с обеих сторон.

Все связи в информационной модели требуют описания, которое, как минимум, включает:

- идентификатор связи;
- формулировку сущности связи;
- вид связи (ее множественность и условность);
- способ описания связи с помощью вспомогательных атрибутов объектов.

Дальнейшее развитие представлений информационного моделирования связано с развитием понятия связи, структур, ими образуемых, и задач, которые могут быть решены на этих структурах. Нам уже известна простая последовательная структура экземпляров - очередь, см. рис. 1.34. Возможными обобщениями информационных моделей являются циклическая структура, таблица (см. табл. 1.10), стек (см. рис. 1.35).

Очень важную роль играет древовидная информационная модель, являющаяся одной из самых распространенных типов классификационных структур. Эта модель строится на основе связи, отражающей отношение части к целому: «А есть часть М» или «М управляет А». Очевидно, древовидная связь является безусловной связью типа один-ко-многим и графически изображена на рис. 1.42, в. На этом же рисунке для сравнения приведены схемы информационных моделей типа «очередь» (а) и «цикл» (б).

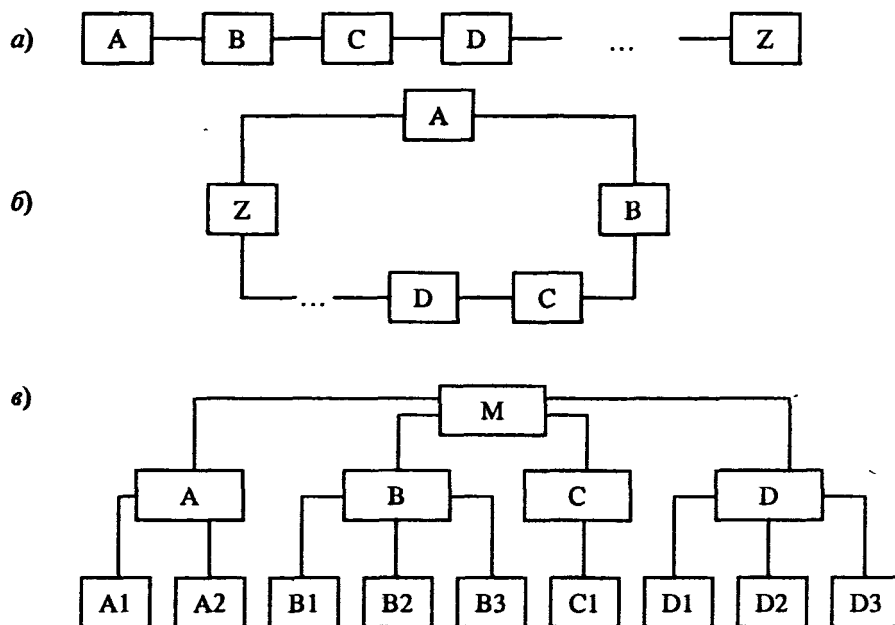


Рис. 1.42. Информационные модели типа «очередь» (а), «цикл» (б), «дерево» (в)

Таким образом, типы данных в программировании, обсуждавшиеся в предыдущем параграфе, тесно связаны с определенными информационными моделями данных.

Еще более общей информационной моделью является, так называемая, графовая структура, рис. 1.43. Графовые структуры являются основой решения огромного количества задач информационного моделирования.

Многие прикладные задачи информационного моделирования были поставлены и изучены достаточно давно, в 50-60-х годах, в связи с активно развивавшимися тогда исследованиями и разработками по научным основам управления в системах различной природы и в связи с попытками смоделировать с помощью компьютеров психическую деятельность человека при решении творческих интеллектуальных задач. Научное знание и модели, которые были получены в ходе решения этих задач, объединены в науке под названием «Кибернетика», в рамках которой существует раздел «Исследования по искусственному интеллекту».

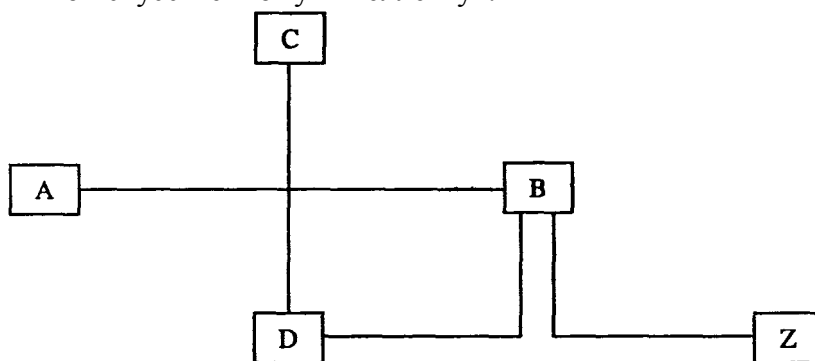


Рис. 1.43. Информационная модель типа «граф»

Контрольные вопросы и задания

1. Что означает понятие «модель» в научном познании?
2. Какие типы моделей существуют?
3. Что такое «информационная модель»?
4. Что такое «объект» с точки зрения информационного моделирования? Какие типы объектов можно различать?
5. Что такое «атрибуты»? Какими они бывают?
6. Что такое «связь»? Какие типы связи различают?
7. Разработайте примеры древовидных структур данных из окружающей реальности.

§ 11. НЕКОТОРЫЕ КИБЕРНЕТИЧЕСКИЕ АСПЕКТЫ ИНФОРМАТИКИ

11.1. ПРЕДМЕТ КИБЕРНЕТИКИ

Слово «кибернетика» происходит от греческого слова, означающего в переводе «кормчий». Его современное значение связано с научной областью, начало которой положила книга американского ученого Норберта Винера «Кибернетика или управление и связь в животном и машине», вышедшая в 1948 г. Вскоре предметом новой науки стали не только биологические и технические системы, но и системы любой природы, способные воспринимать, хранить и перерабатывать информацию и использовать ее для управления и регулирования. В изданной в 1947 г. «Энциклопедии кибернетики» говорится, что это «... наука об общих законах получения, хранения, передачи и преобразования информации в сложных управляющих системах. При этом под управляющими системами здесь понимаются не только технические, а и любые биологические, административные и социальные системы». Таким образом, кибернетика и информатика являются, скорее всего, единой наукой. Сегодня кибернетику все чаще считают частью информатики, ее «высшим» разделом, в какой-то степени аналогичным по положению «высшей математике» по отношению ко всей математике вообще (примерно в таком же положении по отношению к информатике находится и наука «Искусственный интеллект»). Информатика в целом шире кибернетики, так как в информатике имеются аспекты, связанные с архитектурой и программированием ЭВМ, которые непосредственно к кибернетике отнести нельзя.

Кибернетические разделы информатики богаты подходами и моделями в исследовании разнообразных систем и используют в качестве аппарата многие разделы фундаментальной и прикладной математики.

Классическим и до известной степени самостоятельным разделом кибернетики считают **исследование операций**. Под этим термином понимают применение математических методов для обоснования решений в различных областях целенаправленной человеческой деятельности.

Поясним, что понимается под «решением». Пусть предпринимается некоторое мероприятие (в производственной, экономической или социальной сфере), направленное на достижение определенной цели - такое мероприятие называется «операцией». У лица (или группы лиц), ответственного за проведение этого мероприятия, имеется возможность выбора, как его организовать. Например, можно выбрать виды продукции, которые будут выпускаться, оборудование, которое при этом будет применяться, так или иначе распределить имеющиеся средства и т.д. «Операция» есть управляемое мероприятие.

Решение есть выбор из ряда возможностей, имеющихся у ответственного лица. Решения могут быть удачными и неудачными, разумными и неразумными. Оптимальными называют решения, по тем или другим принципам более предпочтительные, чем другие. Цель исследования операций - математическое (количественное) обоснование оптимальных решений.

Исследование операций включает в себя следующие разделы:

1) **математическое программирование** (обоснование планов, программ хозяйственной деятельности); оно включает в себя относительно самостоятельные разделы: **линейное программирование, нелинейное программирование, динамическое программирование** (во всех этих названиях термин «программирование» возник исторически и не имеет отношения к программированию ЭВМ);

2) **теорию массового обслуживания**, опирающуюся на теорию случайных процессов;

3) **теорию игр**, позволяющую обосновывать решения, принимаемые в условиях неполноты информации.

Отметим, что эти разделы не связаны непосредственно с ЭВМ и техническими системами. Иным, быстро развивавшимся в 70-х-80-х годах, разделом кибернетики были системы автоматического (автоматизированного) регулирования. Этот раздел имеет замкнутый, автономный характер, исторически сложившийся самостоятельно. Он тесно связан с разработкой технических систем автоматизированного регулирования и управления технологическими и производственными процессами.

Еще одним классическим разделом кибернетики является распознавание образов, возникшее из задачи моделирования в технических системах восприятия человеком знаков, предметов и речи, а также формирования у человека понятий (обучение в простейшем, техническом смысле). Этот раздел в значительной мере возник из технических потребностей **робототехники**. Например,

требуется, чтобы робот-сборщик распознавал нужные детали. При автоматической сортировке (или отбраковке) деталей необходима способность распознавания.

Вершиной кибернетики (и всей информатики в целом) является раздел, посвященный проблемам **искусственного интеллекта**. Большинство современных систем управления обладают свойством принятия решений - свойством интеллектуальности, т.е. в них смоделирована интеллектуальная деятельность человека при принятии решений.

11.2. УПРАВЛЯЕМЫЕ СИСТЕМЫ

Несмотря на такое многообразие задач, решаемых в разных разделах кибернетики, разнообразие моделей, подходов и методов, кибернетика остается единой наукой благодаря использованию общей методологии, основанной на **теории систем и системном анализе**.

Система - это предельно широкое, начальное, не определяемое строго понятие. Предполагается, что система обладает структурой, т.е. состоит из относительно обособленных частей (элементов), находящихся, тем не менее, в существенной взаимосвязи и взаимодействии. Существенность взаимодействия состоит в том, что благодаря ему элементы системы приобретают все вместе некую новую функцию, новое свойство, которыми не обладает ни один из элементов в отдельности. В этом состоит отличие системы от сети, также состоящей из отдельных элементов, но не связанных между собой существенными отношениями. Сравните, например, предприятие, цеха которого образуют систему, поскольку лишь все вместе приобретают свойство выпускать конечную продукцию (и ни один из них в отдельности с этой задачей не справится), и сеть магазинов, которые могут работать независимо друг от друга.

Кибернетика как наука об управлении изучает не все системы вообще, а только управляемые системы. Зато область интересов и приложений кибернетики распространяется на самые разнообразные биологические, экономические, социальные системы.

Одной из характерных особенностей управляемой системы является возможность переходить в различные состояния под влиянием различных управляющих воздействий. Всегда существует некое множество состояний системы, из которых производится выбор предпочтительного состояния.

Отвлекаясь от конкретных особенностей отдельных кибернетических систем и выделяя общие для некоторого множества систем закономерности, описывающие изменение их состояния при различных управляющих воздействиях, мы приходим к понятию абстрактной **кибернетической системы**. Ее составляющими являются не конкретные предметы, а абстрактные элементы, характеризующиеся определенными свойствами, общими для широкого класса объектов.

Поскольку под кибернетическими системами понимаются управляемые системы, в них должен присутствовать механизм, осуществляющий функции управления. Чаще всего этот механизм реализуется в виде органов, специально предназначенных для управления, рис. 1.44.

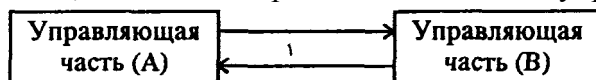


Рис. 1.44. Схематическое изображение кибернетической системы в виде совокупности управляющей (А) и управляемой (В) частей

Стрелками на рисунке обозначены воздействия, которыми обмениваются части системы. Стрелка, идущая от управляющей части системы к управляемой, обозначает сигналы управления. Управляющая часть системы, вырабатывающая сигналы управления, называется управляющим устройством. Управляющее устройство может вырабатывать сигналы управления, обычно на основе информации о состоянии управляемой системы (изображены на рисунке стрелкой от управляемой части системы к управляющей ее части), о требуемом ее состоянии, о возмущающих воздействиях. Совокупность правил, по которым информация, поступающая в управляющее устройство, перерабатывается в сигналы управления, называется алгоритмом управления.

На основе введенных понятий можно определить понятие «управление». Управление - это воздействие на объект, выбранное из множества возможных воздействий на основе имеющейся для этого информации, улучшающее функционирование или развитие данного объекта.

В системах управления решаются четыре основных типа задач управления: 1) регулирование (стабилизация), 2) выполнение программы, 3) слежение и 4) оптимизация.

Задачами регулирования являются задачи поддержания параметров системы -управляемых величин - вблизи некоторых неизменных заданных значений $\{x\}$ несмотря на действие возмущений M , влияющих на значения $\{x\}$. Здесь имеется в виду активная защита от возмущений, принципиально отличающаяся от пассивного способа защиты. **Пассивная защита** заключается в придании объекту таких свойств, чтобы зависимость интересующих нас параметров от внешних возмущений была мала. Примером пассивной защиты является теплоизоляция для поддержания заданной температуры системы, антикоррозионные покрытия деталей машин. **Активная защита** предполагает выработку в управляющих системах управляющих воздействий, противодействующих возмущениям. Так, задача поддержания необходимой температуры системы может быть решена с помощью управляемого подогрева или охлаждения.

Задача выполнения программы возникает в случаях, когда заданные значения управляемых величин $\{x\}$ изменяются во времени известным образом, например, в производстве при выполнении работ согласно заранее намеченному графику. В биологических системах примерами выполнения программы являются развитие организмов из яйцеклеток, сезонные перелеты птиц, метаморфозы насекомых.

Задача слежения - поддержание как можно более точного соответствия некоторого управляемого параметра $X_0(t)$ текущему состоянию системы, меняющемуся непредвидимым образом. Необходимость в слежении возникает, например, при управлении производством товаров в условиях изменения спроса.

Задачи оптимизации - установления наилучшего в определенном смысле режима работы или состояния управляемого объекта - встречаются весьма часто. Примерами являются: управление технологическими процессами с целью минимизации потерь сырья и т.д.

Системы, в которых для формирования управляющих воздействий не используется информация о значениях, которые управляемые величины принимают в процессе управления, называются **разомкнутыми системами управления**. Структура такой системы показана на рис. 1.45.

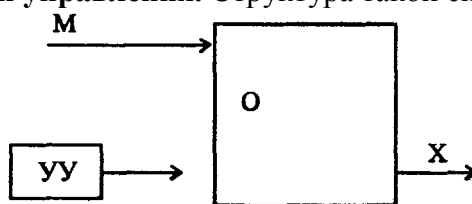


Рис. 1.45. Алгоритм управления, реализуемый управляющим устройством УУ, которое обеспечивает слежение за возмущением M и компенсацию этого возмущения, без использования управляемой величины X

Напротив, в замкнутых системах управления для формирования управляющих воздействий используется информация о значении управляемых величин. Структура такой системы показана на рис. 1.46.

Обратная связь является одним из важнейших понятий кибернетики, помогающим понять многие явления, которые происходят в управляемых системах различной природы. Обратную связь можно обнаружить при изучении процессов, протекающих в живых организмах, экономических структурах, системах автоматического регулирования. Обратная связь, увеличивающая влияние входного воздействия на управляемые параметры системы, называется положительной, уменьшающая влияние входного воздействия - отрицательной.

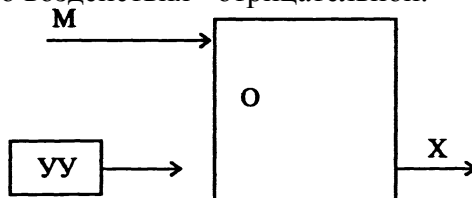


Рис. 1.46. Связь между выходными параметрами A'' и входными Y одного и того же элемента управляемой системы называется обратной связью

Положительная обратная связь используется во многих технических устройствах для усиления, увеличения значений входных воздействий. Отрицательная обратная связь используется для восстановления равновесия, нарушенного внешним воздействием на систему.

11.3. ФУНКЦИИ ЧЕЛОВЕКА И МАШИНЫ В СИСТЕМАХ УПРАВЛЕНИЯ

Хорошо изученной сферой применения кибернетических методов является технологическая и производственная сфера, управление промышленным предприятием. Задачи, возникающие в управлении предприятием среднего и большого масштаба, уже весьма сложны, но допускают решение с использованием электронно-вычислительных машин. Системы управления хозяйством предприятий или территорий (регионов, городов), использующие ЭВМ для переработки и хранения информации, получили название автоматизированных систем управления (АСУ). По своему характеру такие системы являются человеко-машинными, т.е. наряду с использованием мощных компьютеров предполагающими наличие в них человека с его естественным интеллектом. В человеко-машинных системах предполагается следующее разделение функций человека и машины: **машина** хранит и перерабатывает большие массивы информации, осуществляет информационное обеспечение принятия решений человеком; **человек** принимает управленческие решения.

Чаще в человеко-машинных системах компьютеры выполняют рутинную, нетворческую, трудоемкую переработку информации, освобождая человеку время для творческой деятельности. Однако целью развития компьютерной (информационной) технологии управления является полная автоматизация деятельности, включающая частичное или полное освобождение человека от необходимости принятия решений. Это связано не только со стремлением разгрузить человека, но и с тем, что развитие техники и технологий привело к ситуациям, когда человек в силу присущих ему физиологических и психических ограничений просто не успевает принимать решения в реальном масштабе времени протекания процесса, что грозит катастрофическими последствиями. Примеры - необходимость включения аварийной защиты ядерного реактора, реакция на события, происходящие при запусках космических аппаратов и т.д.

Система, заменяющая человека, должна будет обладать интеллектом, в какой-то мере подобным человеческому - **искусственным интеллектом**. Исследовательское направление в области систем искусственного интеллекта также относится к кибернетике, однако вследствие его важности для перспектив всей информатики в целом мы рассмотрим его в отдельном параграфе.

Контрольные вопросы и задания

1. Каков предмет науки «Кибернетика»?
2. Охарактеризуйте задачи, решаемые в научном разделе «исследование операций».
3. Какое место в кибернетике занимает теория автоматического управления и регулирования?
4. Что означает понятие «система»?
5. Что такое «система управления»?
6. Охарактеризуйте задачи, возникающие в системах управления.
7. Что такое «обратная связь»? Приведите примеры обратной связи в окружающих вас управляемых системах.
8. Что такое АСУ?
9. Каково место человека и ЭВМ в человеко-машинных системах управления?

§ 12. ПОНЯТИЕ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

12.1. НАПРАВЛЕНИЯ ИССЛЕДОВАНИЙ И РАЗРАБОТОК В ОБЛАСТИ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Научное направление, связанное с машинным моделированием человеческих интеллектуальных функций - искусственный интеллект - возникло в середине 60-х годов XX столетия. Его возникновение непосредственно связано с общим направлением научной и инженерной мысли, которое привело к созданию компьютера - направлением на автоматизацию человеческой интеллектуальной деятельности, на то, чтобы сложные интеллектуальные задачи, считавшиеся прерогативой человека, решались техническими средствами.

Говоря о сложных интеллектуальных задачах следует понимать, что всего 300 -400 лет назад перемножение больших чисел вполне относилось к таковым; однако, усвоив в детстве правило умножения столбиком, современный человек пользуется им не задумываясь, и вряд ли эта

задача сегодня является «сложной интеллектуальной». По-видимому, в круг таковых следует включить те задачи, для решения которых нет «автоматических» правил, т.е. нет алгоритма (пусть даже и очень сложного), следование которому всегда приводит к успеху. Если для решения задачи, которая нам сегодня представляется относящейся к указанному кругу, в будущем придумают четкий алгоритм, она перестанет быть «сложной интеллектуальной».

Несмотря на свою краткость, история исследований и разработок систем искусственного интеллекта может быть разделена на четыре периода:

- 60-е - начало 70-х годов XX века - исследования по «общему интеллекту», попытки смоделировать общие интеллектуальные процессы, свойственные человеку: свободный диалог, решение разнообразных задач, доказательство теорем, различные игры (типа шашек, шахмат и т.д.), сочинение стихов и музыки и т.д.;

- 70-е годы - исследования и разработка подходов к формальному представлению знаний и умозаключений, попытки свести интеллектуальную деятельность к формальным преобразованиям символов, строк и т.д.;

- с конца 70-х годов - разработка специализированных на определенных предметных областях интеллектуальных систем, имеющих прикладное практическое значение (экспертных систем);

- 90-е годы - фронтальные работы по созданию ЭВМ 5-го поколения, построенных на иных принципах, чем обычные универсальные ЭВМ, и программного обеспечения для них.

В настоящее время «искусственный интеллект» - мощная ветвь информатики, имеющая как фундаментальные, чисто научные основы, так и весьма развитые технические, прикладные аспекты, связанные с созданием и эксплуатацией работоспособных образцов интеллектуальных систем. Значение этих работ для развития информатики таково, что именно от их успеха зависит появление ЭВМ нового 5-го поколения. Именно этот качественный скачок возможностей компьютеров - обретение ими в полной мере интеллектуальных возможностей - положен целью развития вычислительной техники в ближайшей перспективе и является признаком компьютерной техники нового поколения.

Любая задача, для которой не известен алгоритм решения, может быть отнесена к сфере искусственного интеллекта. Примерами могут быть игра в шахматы, медицинская диагностика, составление резюме текста или перевода его на иностранный язык - для решения этих задач не существует четких алгоритмов. Еще две характерные особенности задач искусственного интеллекта: преобладающее использование информации в символьной (а не в числовой) форме и наличие выбора между многими вариантами в условиях неопределенности.

Перечислим отдельные направления, где применяются методы искусственного интеллекта.

1. Восприятие и распознавание образов (задача, упоминавшаяся ранее, как одно из направлений кибернетики). Теперь под этим понимаются не просто технические системы, воспринимающие визуальную и звуковую информацию, кодирующие и размещающие ее в памяти, проблемы понимания и логического рассуждения в процессе обработки визуальной и речевой информации.

2. Математика и автоматическое доказательство теорем.

3. Игры. Как и формальные системы в математике, игры, характеризующиеся конечным числом ситуаций и четко определенными правилами, с самого начала исследований по искусственному интеллекту привлекли к себе внимание как предпочтительные объекты исследования, полигон для применения новых методов. Интеллектуальными системами был быстро достигнут и превзойден уровень человека средних способностей, однако уровень лучших специалистов не достигнут до сих пор. Возникшие трудности оказались характерными и для многих других ситуаций, так как в своих «локальных» действиях человек использует весь объем знаний, который он накопил за всю свою жизнь.

4. Решение задач. В данном случае понятие «решение» используется в широком смысле, относится к постановке, анализу и представлению конкретных ситуаций, а рассматриваемые задачи - те, которые встречаются в повседневной жизни, для решения которых требуется изобретательность и способность к обобщению.

5. Понимание естественного языка. Здесь ставится задача анализа и генерации текстов, их внутреннего представления, выявление знаний, необходимых для понимания текстов. Трудности связаны, в частности, с тем, что значительная часть информации в обычном диалоге не выражается определенно и ясно. Предложениям естественного языка присуща:

- неполнота;
- неточность;
- нечеткость;
- грамматическая некорректность;
- избыточность;
- зависимость от контекста;
- неоднозначность.

Однако такие свойства языка, являющегося результатом многовекового исторического развития, служат условием функционирования языка как универсального средства общения. Вместе с тем, понимание предложений естественного языка техническими системами с трудом поддается моделированию из-за этих особенностей языка (да и вопрос о том, что такое «понимание», нуждается в размышлениях). В технических системах должен использоваться формальный язык, смысл предложений которого однозначно определяется их формой. Перевод с естественного языка на формальный является нетривиальной задачей.

6. Выявление и представление знаний экспертов в экспертных системах. Экспертные системы - интеллектуальные системы, вобравшие в себя знания специалистов в конкретных видах деятельности - имеют большое практическое значение, с успехом применяются во многих областях, таких как автоматизированное проектирование, медицинская диагностика, химический анализ и синтез и т.д.

Во всех этих направлениях главные трудности связаны с тем, что недостаточно изучены и поняты принципы человеческой интеллектуальной деятельности, процесс принятия решений и решение задач. Если в 60-х годах широко обсуждался вопрос «может ли компьютер мыслить», то теперь вопрос ставится иначе: «достаточно ли хорошо человек понимает, как он мыслит, чтобы передать эту функцию компьютеру»? В силу этого, работы в области искусственного интеллекта тесно соприкасаются с исследованиями по соответствующим разделам психологии, физиологии, лингвистики.

12.2. ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В СИСТЕМАХ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Основной особенностью интеллектуальных систем является то, что они основаны на знаниях, а вернее, на некотором их представлении. **Знания** здесь понимаются как хранимая (с помощью ЭВМ) информация, формализованная в соответствии с некоторыми правилами, которую ЭВМ может использовать при логическом выводе по определенным алгоритмам. Наиболее фундаментальной и важной проблемой является описание смыслового содержания проблем самого широкого диапазона, т.е. должна использоваться такая форма описания знаний, которая гарантировала бы правильную обработку их содержимого по некоторым формальным правилам. Эта проблема называется проблемой представления знаний.

В настоящее время наиболее известны три подхода к представлению знаний в обсуждаемых системах:

- продукционная и логическая модели;
- семантические сети;
- фреймы.

Продукционные правила - наиболее простой способ, представления знаний. Он основан на представлении знаний в форме правил, структурированных в соответствии с образцом «ЕСЛИ - ТО». Часть правила «ЕСЛИ» называется посылкой, а «ТО» - выводом или действием. Правило в общем виде записывается так:

ЕСЛИ $A_1, A_2, \dots, A_n$, ТО B .

Такая запись означает, что «если все условия от A_1 до A_n являются истинными, то B также истинно» или «когда все условия от A_1 до A_n выполняются, то следует выполнить действие B ».

Рассмотрим правило

ЕСЛИ	(1)	у является отцом x
	(2)	z является братом y

ТО

z является дядей x

В данном случае число условий $n = 2$.

В случае $n = 0$ продукция описывает знание, состоящее только из вывода, т.е. факт. Примером такого знания является факт «атомная масса железа 55,847 а.е.м».

Переменные x , y и z показывают, что правило содержит некое универсальное, общее знание, абстрагированное от конкретных значений переменных. Одна и та же переменная, использованная в выводе и различных посылках, может получать различные конкретные значения.

Знания, представленные в интеллектуальной системе, образуют **базу знаний**. В интеллектуальную систему входит также **механизм выводов**, который позволяет на основе знаний, имеющихся в базе знаний, получать новые знания.

Проиллюстрируем сказанное. Положим, что в базе знаний вместе с описанным выше правилом содержатся и такие знания:

ЕСЛИ (1) z является отцом x
(2) z является отцом y
(3) x и y не являются одним и тем же человеком
ТО x и y являются братьями
Иван является отцом Сергея
Иван является отцом Павла
Сергей является отцом Николая

Из представленных знаний можно формально вывести заключение, что Павел является дядей Николая. При этом считается, что одинаковые переменные, входящие в разные правила, независимы; объекты, имена которых эти переменные могут получать, никак не связаны между собой. Формализованная процедура, использующая сопоставление (при котором устанавливается, совпадают ли между собой две формы представления, включая подстановку возможных значений переменных), поиск в базе знаний, возврат к исходному состоянию при неудачной попытке решения, представляет собой механизм выводов.

Простота и наглядность представления знаний с помощью продукций обусловила его применение во многих системах, которые называются продукционными.

Семантическая сеть - иной подход к представлению знаний, который основан на изображении понятий (сущностей) с помощью точек (узлов) и отношений между ними с помощью дуг на плоскости. Семантические сети способны отображать структуру знаний во всей сложности их взаимосвязей, увязать в единое целое объекты и их свойства. В качестве примера может быть приведена часть семантической сети, относящейся к понятию «фрукты» (рис. 1.47).

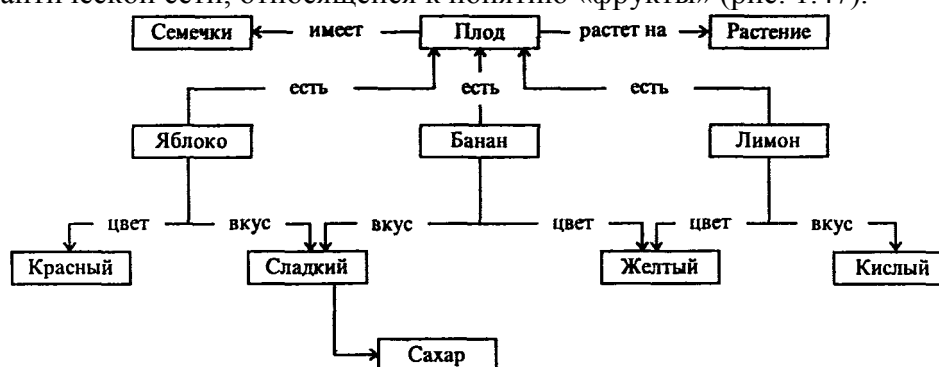


Рис. 1.47. Пример семантической сети

Фреймовая система имеет все свойства, присущие языку представления знаний, и одновременно является новым способом обработки информации. Слово «фрейм» в переводе с английского языка означает «рамка». Фрейм является единицей представления знаний об объекте, которую можно описать некоторой совокупностью понятий и сущностей. Фрейм имеет определенную внутреннюю структуру, состоящую из множества элементов, называемых **слотами**. Каждый слот, в свою очередь, представляется определенной структурой данных, процедурой, или может быть связан с другим фреймом.

Фрейм: человек

Класс	:	Животное
Структурный элемент	:	Голова, шея, руки, ноги,...
Рост	:	30-220 см
Масса	:	1 - 200 кг
Хвост	:	Нет
Фрейм аналогии	:	Обезьяна

Существуют и другие, менее распространенные подходы к представлению знаний в интеллектуальных системах, в том числе гибридные, на основе уже описанных подходов.

Перечислим главные особенности машинного представления данных.

1. Внутренняя интерпретируемость. Обеспечивается наличием у каждой информационной единицы своего уникального имени, по которому система находит ее для ответа на запросы, в которых это имя упомянуто.

2. Структурированность. Информационные единицы должны обладать гибкой структурой, для них должен выполняться «принцип матрешки», т.е. вложенности одних информационных единиц в другие, должна существовать возможность установления соотношений типа «часть - целое», «род - вид», «элемент - класс» между отдельными информационными единицами.

3. Связность. Должна быть предусмотрена возможность установления связей различного типа между информационными единицами, которые бы характеризовали отношения между информационными единицами. Эти отношения могут быть как декларативными (описательными), так и процедурными (функциональными).

4. Семантическая метрика. Позволяет устанавливать ситуационную близость информационных единиц, т.е. величину ассоциативной связи между ними. Такая близость позволяет выделять в знаниях некоторые типовые ситуации, строить аналогии.

5. Активность. Выполнение действий в интеллектуальной системе должно инициироваться не какими-либо внешними причинами, а текущим состоянием представленных в системе знаний. Появление новых фактов или описание событий, установление связей должны стать источником активности системы.

12.3. МОДЕЛИРОВАНИЕ РАССУЖДЕНИЙ

Рассуждение - один из важнейших видов мыслительной деятельности человека, в результате которого он формулирует на основе некоторых предложений, высказываний, суждений новые предложения, высказывания, суждения. Действительный механизм рассуждений человека остается пока недостаточно исследованным. Человеческим рассуждениям присущи: неформальность, нечеткость, нелогичность, широкое использование образов, эмоций и чувств, что делает чрезвычайно трудными их исследование и моделирование. К настоящему времени лучше всего изучены логические рассуждения и разработано много механизмов дедуктивных выводов, реализованных в различных интеллектуальных системах, основанных на представлении знаний с помощью логики предикатов 1-го порядка.

Предикат - это конструкция вида $P(t_1, t_2, \dots, t_n)$, выражающая какую-то связь между некоторыми объектами или свойствами объектов. Обозначение этой связи или свойства, P , называют «предикатным символом»; $t_1, t_2, \dots, t_n$ обозначают объекты, связанные свойством (предикатом) P и называют термами.

Термы могут быть только трех следующих типов:

- 1) константа (обозначает индивидуальный объект или понятие);
- 2) переменная (обозначает в разное время различные объекты);
- 3) составной терм - функция $f(t_1, t_2, \dots, t_n)$, имеющая в качестве своих аргументов m термов $t_1, t_2, \dots, t_m$.

Примеры.

1. Предложение «Волга впадает в Каспийское море» можно записать в виде предиката

впадает (Волга, Каспийское море).

«Впадает» - предикатный символ; «Волга» и «Каспийское море» - термы-константы. Мы могли обозначить отношение «впадает» и объекты «Волга» и «Каспийское море» символами.

Вместо термов-констант можно рассматривать переменные:

впадает (X , Каспийское море)

или

впадает (X, Y).

Это тоже предикаты.

2. Отношение $x + 1 < y$ можно записать в виде предиката $A(x, y)$. Предикатный символ A здесь обозначает то, что останется от $x + 1 < y$, если выбросить из этой записи переменные x и y .

Итак, предикат - это логическая функция, принимающая значения «истина» или «ложь» в зависимости от значений своих аргументов. Количество аргументов y предиката называют его **арностью**.

Так, для наших примеров предикат «впадает» имеет арность 2 и при $X = \text{«Волга»}$, а $Y = \text{«Каспийское море»}$ истинен, а при $X = \text{«Дон»}$, $Y = \text{«Бискайский залив»}$ ложен. Предикат A в примере 2 также имеет арность 2, истинен при $X = 1$, $Y = 3$ и ложен при $X=3$, $Y=1$.

Предикаты могут быть объединены в формулы с помощью логических связок (союзов): $\wedge$ («и», конъюнкция), $\vee$ («или», дизъюнкция), $\sim$ («не», отрицание), $\rightarrow$ («следует», импликация), $\leftrightarrow$ («тогда, и только тогда, когда», эквиваленция).

Ниже приведены таблицы истинности этих союзов, позволяющие определить, истинно или ложно значение формулы-связки при различных значениях, входящих в нее предикатов A и B .

Математически строго формулы логики предикатов определяются рекурсивно:

- 1) предикат есть формула;
- 2) если A и B - формулы, то $A, A \wedge B, A \vee B, A \rightarrow B, A \leftrightarrow B$ - тоже формулы;
- 3) других формул не бывает.

Многие формулы логики предикатов требуют использования **кванторов**, определяющих область значений переменных - аргументов предикатов. Используются кванторы общности (перевернутое A от английского «All» - все) и квантор существования - (перевернутое E от английского «Exists» - «существует»). Запись $\forall x$ читается «для любого x », «для каждого x »; $\exists x$ - «существует x », «хотя бы для одного x ». Кванторы связывают переменные предикатов, на которые они действуют, и превращают предикаты в высказывания.

Таблица 1.12 Истинность связок предикатов (И - истина, Л - ложь)

A	B	$A \wedge B$	$A \vee B$	$\sim A$	$A \rightarrow B$	$A \leftrightarrow B$
И	И	И	И	Л	И	И
И	Л	Л	И	Л	Л	Л
Л	И	Л	И	И	И	Л
Л	Л	Л	Л	И	И	И

Пример.

Введем обозначения: $A(x)$ - студент x учится отлично; $B(x)$ - студент x получает повышенную стипендию. Теперь формула A (Иванов) $\rightarrow B$ (Иванов) означает: студент Иванов учится отлично, следовательно, студент Иванов получает повышенную стипендию, а формула с квантором общности $(\forall x) (A(x) \rightarrow B(x))$ означает: каждый студент, который учится отлично, получает повышенную стипендию.

Из всевозможных формул нам потребуется только один их вид, называемый **фразами Хорна**. Фразы Хорна содержат в общем случае импликацию и конъюнкцию предикатов $A, B_1, B_2, \dots, B_n$ следующим образом: $B_1, B_2, \dots, B_n \rightarrow A$, или в более удобных обозначениях:

$A :- B1, B2, \dots Bn$

(читается: A если $B1$ и $B2$ и ... и Bn).

Очевидно, фраза Хорна является формой записи некоего правила, и в дальнейшем будет называться правилом. Предикат A называется заголовком или головой правила, а предикаты $B1, B2, \dots, Bn$ - его подцелями.

Очевидно, что отдельный предикат является частным случаем фразы Хорна: A .

Другой частный случай фразы Хорна - правило без головы

$:-B1, B2, \dots, Bn,$

или

$:-B.$

Такая фраза Хорна называется вопросом. Мы будем записывать «: - B » в виде «? - B », а «:- $B1, B2, \dots, Bn$ » в виде «? - $B1, B2, \dots, Bn$ ».

Поясним логический смысл такой формулы. Напомним, что импликация $A :- B$ ($B \rightarrow A$) может быть выражена через отрицание и дизъюнкцию: $\sim B \vee A$ (проверьте это с помощью таблицы истинности). Значит, если отбросить A , останется только $\sim B$ - отрицание B . Формула $\leftarrow B1, \dots, Bn$ означает отрицание конъюнкции $\sim\{B1 \wedge B2 \wedge \dots \wedge Bn\}$, что по закону де Моргана $\sim(X \wedge Y) = (\sim X) \vee (\sim Y)$ равно $(\sim B1) \vee (\sim B2) \vee \dots \vee (\sim Bn)$ - дизъюнкции отрицаний.

Множество фраз Хорна применительно к некоторой проблемной области образует теорию (в логическом смысле).

Пример.

Рассмотрим предметную область: сдачу экзамена по некоторой дисциплине. Введем обозначения:

A - студент успешно сдает экзамен.

B - студент посещал занятия.

C - студент освоил учебный материал.

D - студент занимался самостоятельно.

E - студент подготовил шпаргалку.

Введем знания о предметной области:

Студент успешно сдаст экзамен, если студент освоил учебный материал.

Студент освоил учебный материал, если студент посещал занятия и студент занимался самостоятельно. Студент посещал занятия. Студент занимался самостоятельно.

Форма логической записи:

$A :- C;$

$C :- B, D;$

$B;$

$D.$

В приведенном примере можно выполнить логический вывод. Так, из истинности фактов B и D и правила $C :- B, D$ следует истинность C , и из правила $A :- C$ - истинность предиката A , т.е., студент успешно сдаст экзамен. Кроме того, правила $A :- C$ и $C :- B, D$ можно было бы переписать в виде $A :- B, D$.

В этих случаях используют правила вывода, называемые **методом резолюций**.

Рассмотрим простейшую форму резолюции. Допустим, имеются «родительские» предложения

отрицание: $\sim A$

импликация: $A :- B.$

В результате одного шага резолютивного вывода получаем новое предложение: B , которое называется **резольвентой**. В этом случае резолюция соответствует стандартному пропозициональному правилу вывода:

допуская, что не	A
и	A , если B
выводим	не B .

Еще более простой случай:

отрицание:	$\sim A$
факт:	A .
Резольвента - противоречие.	

В общем случае имеются родительские предложения

$\sim(A_1 \wedge \dots \wedge A_n)$
 $A_k: \neg B_1, \dots, B_m \quad 1 \leq k \leq n.$

В качестве резольвенты за один шаг вывода получается $\sim(A_1 \wedge \dots \wedge A_{k-1} \wedge B_1 \wedge \dots \wedge B_m \wedge A_{k+1} \wedge \dots \wedge A_n)$.

Таким образом, резолюция является подстановкой предикатов - подцелей $B_1, \dots, B_m$, вместо соответствующего предиката A_k из отрицания. Отрицание инициирует логический вывод и поэтому называется запросом (или вопросом) и обозначается $A_1, A_2, \dots, A_n$. Смысл метода резолюций состоит в том, что строится отрицание конъюнкции и проверяется, истинно его значение или ложно. Если значение результирующей конъюнкции ложь, значит, получилось противоречие и, поскольку на старте было отрицание предикатов, выполнено доказательство «от обратного». Если получено значение «истина», то доказательство не выполнено.

Пример.

Пусть предикат $дает(X, Y, Z)$ означает, что « X дает Y некоторому объекту Z », а предикат $получают(X, Y)$ означает, что « Y получает Z ». Пусть знания об этих отношениях выражаются предложениями

- 1) $получает(вы, сила)$:- дает (логика, сила, вы);
- 2) $дает$ (логика, сила, вы).

Задача, которую нужно решить, состоит в том, чтобы ответить на вопрос: получаете ли вы силу?

Представим этот вопрос в виде отрицания $\neg пол\!учает(вы, сила)$. Резолюция предложения 1 и отрицания приводит к $\neg дает$ (логика, сила, вы), что вместе с фактом 2 приводит к противоречию. Следовательно, ответом исходной задачи является «да».

Пока что мы рассмотрели резолюцию для высказываний или предикатов без переменных. Если же вывод производится для множества предикатов с переменными в качестве аргументов, эти переменные в ходе вывода получают значения соответствующих констант или, как еще говорят, конкретизируются константами.

Рассмотрим это на примере.

Пример.

Рассмотрим следующие родительские предложения:

- 1) $\neg пол\!учает$ (вы, Y);
- 2) $пол\!учает$ (X , сила) :- $дает$ (Z , сила, X).

Они содержат три переменные X, Y и Z , которые неявно находятся под действием квантора общности. Так, предложение 1 утверждает, что «для всех Y вы не получаете Y », а 2 - «для всех Z любой X получает силу, если Z дает силу X ». Правило резолюции требует совпадения предиката из отрицания 1 и головы правила 2. Это означает, что переменные получают значения (конкретизируются) соответственно их месту в предложениях 1 и 2 следующим образом: $X = вы$, $Y = сила$.

Предикат *получает* (вы, сила) называется общим примером для предикатов *получает*(вы Y) и *получает* (X, сила).

Изложенные положения логики предикатов находят реализацию и дальнейшее развитие в языке программирования Пролог.

12.4. ИНТЕЛЛЕКТУАЛЬНЫЙ ИНТЕРФЕЙС ИНФОРМАЦИОННОЙ СИСТЕМЫ

Анализ развития средств вычислительной техники позволяет утверждать, что она постоянно эволюционирует в двух направлениях. Первое из них связано с улучшением параметров существующих компьютеров, повышением их быстродействия, увеличением объемов их оперативной и дисковой памяти, а также с совершенствованием и модификацией программных средств, ориентированными на повышение эффективности выполнения ими своих функций. Это можно назвать развитием по горизонтали.

Второе направление определяет изменения в технологии обработки информации, приводящие к улучшению использования компьютерных систем. Развитие в этом направлении связано с появлением новых типов компьютеров и качественно новых программных средств, дополняющих уже существующие. Такое развитие можно назвать развитием по вертикали.

Развитие программных средств идет по пути увеличения их дружелюбности, т.е. такого упрощения управления ими, что от пользователя не требуется специальной подготовки, и система создает максимально комфортные условия для его работы. Основным ориентир в совершенствовании вычислительных систем - превращение их в удобного партнера конечного пользователя при решении задач в ходе его профессиональной деятельности.

Для обеспечения наибольшей дружелюбности интерфейса программного средства с пользователем первый должен стать интеллектуальным. **Интеллектуальный интерфейс**, обеспечивающий непосредственное взаимодействие конечного пользователя и компьютера при решении задачи в составе человеко-машинной системы, должен выполнять три группы функций:

- обеспечение для пользователя возможности постановки задачи для ЭВМ путем сообщения только ее условия (без задания программы решения);
- обеспечение для пользователя возможности формирования сред решения задачи с использованием только терминов и понятий из области профессиональной деятельности пользователя, естественных форм представления информации;
- обеспечение гибкого диалога с использованием разнообразных средств, в том числе не регламентируемых заранее, с коррекцией возможных ошибок пользователя.

Структура системы (рис. 1.48), удовлетворяющей требованиям новой технологии решения задач, состоит из трех компонент:

- **исполнительной системы**, представляющей собой совокупность средств, обеспечивающих выполнение программ;
- **базы знаний**, содержащей систему знаний о проблемной среде;
- **интеллектуального интерфейса**, обеспечивающего возможность адаптации вычислительной системы к пользователю.

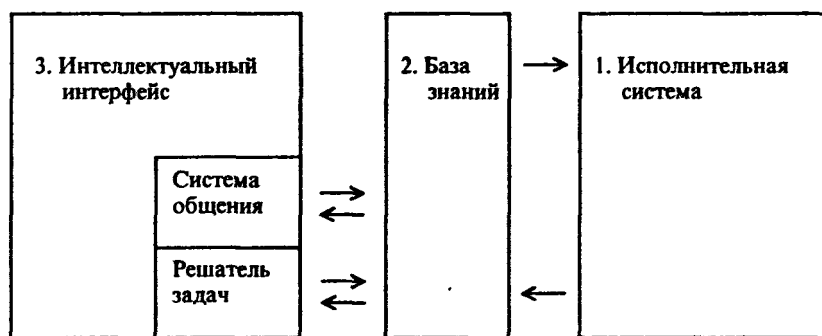


Рис. 1.48. Структура современной системы решения прикладных задач

Такая система существенно отличается от создававшихся на более ранних этапах развития информатики и вычислительной техники. Путь реализации новейших информационных технологий предполагает использование вычислительных систем, построенных на основе представления

знаний предметной области задачи и интеллектуального интерфейса.

12.5. СТРУКТУРА СОВРЕМЕННОЙ СИСТЕМЫ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

Разработки систем искусственного интеллекта шли сначала по пути моделирования общих интеллектуальных функций индивидуального сознания. Однако, развитие вычислительной техники и программного обеспечения в 90-х годах опровергает прогнозы предыдущих десятилетий о скором переходе к ЭВМ 5-го поколения. Интеллектуальные функции основной массы программных систем общения на естественном языке пока не находят широкого внедрения в промышленных масштабах.

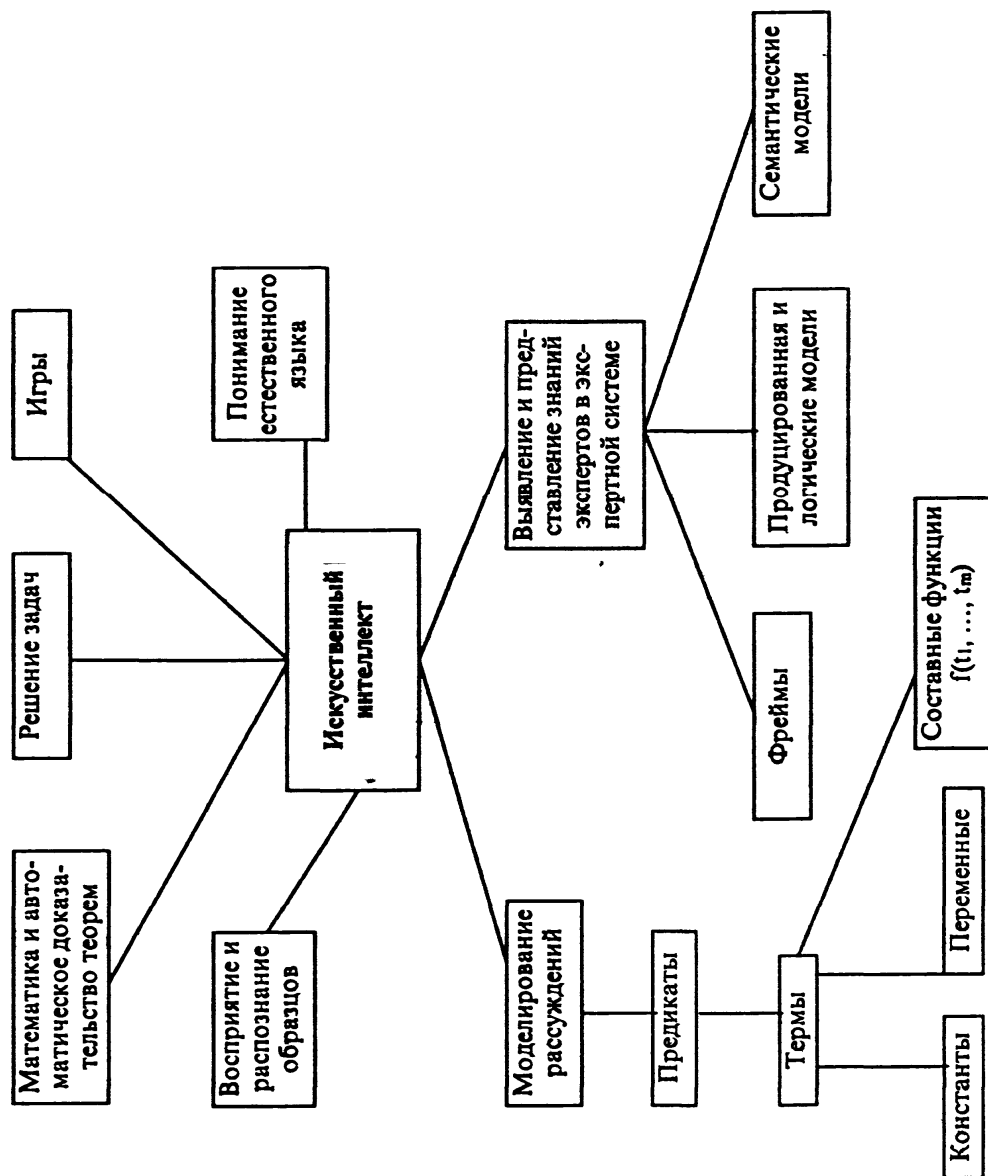
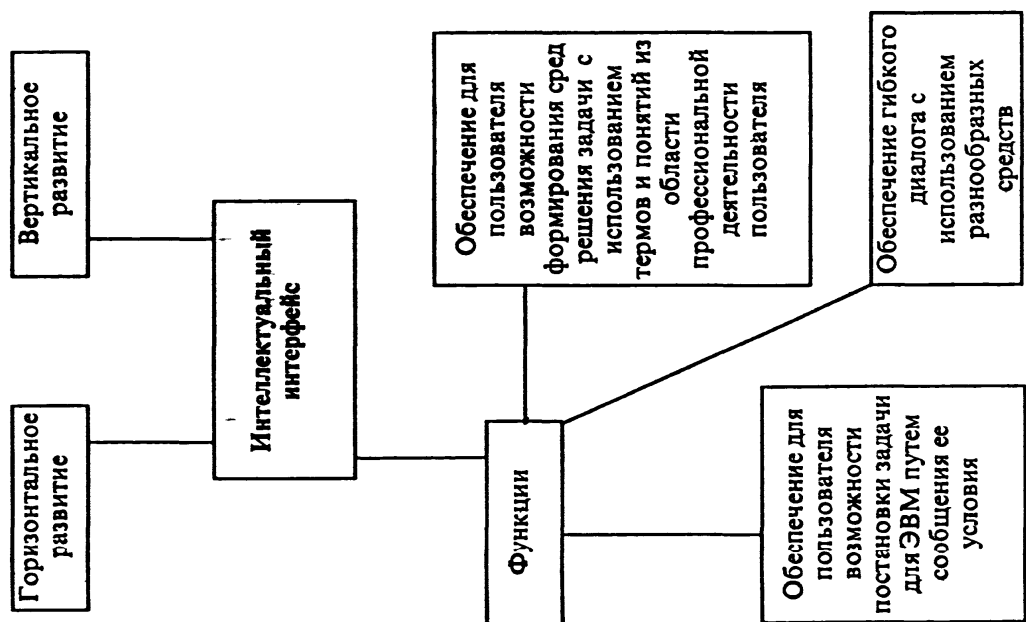
Характерную инфляцию претерпело такое понятие, как «новая информационная технология». Первоначально это понятие означало интеллектуальный интерфейс к базе данных, позволяющий прикладным пользователям общаться с ней непосредственно на естественном языке. Ныне под «новыми информационными технологиями» понимают просто технологии, существенно использующие вычислительную технику в обработке информации, в том числе основанные на применении текстовых и табличных процессоров, а также информационных систем.

Столкнувшись с непреодолимыми проблемами, разработчики систем, обладающих «общим» искусственным интеллектом, пошли по пути все большей и большей специализации, вначале по направлению к экспертным системам, затем - к отдельным очень специфичным интеллектуальным функциям, встроенным в инструментальные программные средства, не считавшиеся до настоящего времени сферой разработок по искусственному интеллекту. Например, такие системы сейчас часто обладают возможностями аналитических математических вычислений, перевода технических и деловых текстов, распознавания текста при вводе сканером, синтаксического анализа фраз и предложений, самонастраиваемостью и т.д.

Парадигма исследований и разработок в области искусственного интеллекта постепенно пересматривается. По-видимому, возможности скорого развития программных систем, моделирующих интеллектуальные функции индивидуального сознания, в значительной мере исчерпаны. Необходимо обратить внимание на новые возможности, которые открывают в отношении общественного сознания информационные системы и сети. Развитие вычислительных систем и сетей ведет по-видимому, к созданию нового типа общественного сознания, в которое информационные средства будут органично встроены как технологическая среда обработки и передачи информации. После этого человечество получит именно гибридный человеко-машинный интеллект не столько в масштабе индивидуального сознания, сколько в сфере социальной практики.

Контрольные вопросы и задания

1. Какова теория возникновения и развития исследований по искусственному интеллекту?
2. Каковы отличительные черты задач из сферы искусственного интеллекта?
3. Охарактеризуйте направления исследований по искусственному интеллекту.
4. Что такое «знания» с точки зрения систем искусственного интеллекта?
5. В чем состоит метод представлений знаний с помощью продукций?
6. На чем основано представление знаний с помощью семантической сети?
7. Как фреймовые системы могут использоваться для представления знаний?
8. В чем отличия представления знаний в интеллектуальных системах от представления просто данных?
9. Что значит понятие «предикат»?
10. Что такое «фраза Хорна»?
11. Как происходит логический вывод с помощью метода резолюций?
12. В каком направлении развиваются интерфейсные части информационных систем?
13. В чем состоит принцип дружественности программных средств?
14. Какова структура перспективных информационных систем будущего?



Дополнительная литература к главе 1

1. *Абрамов Ю. Ф.* Картина мира и информация (философские очерки). - Иркутск: ИГУ,

1988.

2. Абдеев Р. Ф. Философия информационной цивилизации. ~ М.: ВЛАДОС, 1994.
3. Аиламазян А. К., Стась Е.В. Информатика и теория развития. - М.: Наука, 1992.
4. Ахундов М.Д. Эволюция и смена научных картин мира. В кн.: Философия, естествознание, социальное развитие. - М.: Наука, 1989.
5. Брой М. Информатика. Основополагающее введение. В 3-х частях. - М.: Диалог-МИФИ, 1996.
6. Будущее искусственного интеллекта. - М.: Наука, 1991.
7. Вейценбаум Дж. Возможности вычислительных машин и человеческий разум. -М.: Радио и связь, 1982.
8. Венда В. Ф. Системы гибридного интеллекта: эволюция, психология, информатика. - М.: Машиностроение, 1990.
9. Вирт Н. Алгоритмы и структуры данных. - М.: Мир, 1989.
10. Вирт Н. Алгоритмы + структура данных = программы. - М.: Мир, 1985.
11. Глинский Б. А. Философские и социальные аспекты информатики. - М.: Наука, 1990.
12. Глушков В. М. Основы безбумажной информатики. /Изд. 2-е. - М.: Наука, 1987.
13. Глушков В. М. Кибернетика. Вопросы теории и практики. - М.: Наука, 1986.
14. Голицын Г. А., Петров В. М. Информация, поведение, творчество. - М.: Наука, 1991.
15. Гриценко В. И., Паньшин Б.Н. Информационная технология: вопросы развития и применения. - Киев: Наукова Думка, 1988.
16. Громов Г. Р. Национальные информационные ресурсы. Проблемы промышленной эксплуатации. - М.: Наука, 1985.
17. Громов Г. Р. Очерки информационной технологии. - М.: Инфоарт, 1992.
18. Дмитриев В. И. Прикладная теория информации. - М.: Высшая школа, 1989.
19. Интеллектуальные процессы и их моделирование. - М.: Наука, 1987.
20. Информатизация общества и бизнес. - М.: ИНИОН, 1992.
21. Информатика и культура. - М. Наука, 1990.
22. Информационная революция: наука, экономика, технология. Серия «Информация. Наука. Общество». - М.: ИНИОН, 1993.
23. Искусственный интеллект: - В 3-х кн. Кн.3. Программные и аппаратные средства: Справочник / Под ред В.Н.Захарова, В.Ф.Хорошевского. - М.: Радио и связь, 1990.
24. Каныгин Ю. М., Калинич Г. И. Основы теоретической информатики. - Киев: Наукова Думка, 1990.
25. Когнитивная психология и искусственный интеллект. Серия «Информация. Наука. Общество». - М.: ИНИОН, 1993.
26. Лорьер Ж.-Л. Системы искусственного интеллекта. - М.: Финансы и статистика, 1982.
27. Марков М. Технология и эффективность социального управления. - М.: Прогресс, 1982.
28. Матросов В.Л. Теория алгоритмов. - М.: Прометей, 1989.
29. Минский М. Вычисления и автоматы. - М.: Мир, 1971.
30. Новик И. Б., Абдуллаев Ф.Ш. Введение в информационный мир. - М.: Наука, 1991.
31. Перспективы информатизации общества. Ч.1,2. Серия «Информация. Наука. Общество». - М.: ИНИОН, 1990.
32. Першиков В. И., Савинков В. М. Толковый словарь по информатике. - М.: Финансы и статистика, 1991.
33. Попов Э.П. Экспертные системы: Решение неформализованных задач в диалоге с ЭВМ. - М.: Наука, 1987.
34. Поспелов Г. С. Искусственный интеллект - основа новой информационной технологии. - М.: Наука, 1988.
35. Рахитов А. И. Философия компьютерной революции. - М.: ИПЛ, 1991.
36. Свириденко С. С. Современные информационные технологии. - М.: Радио и связь, 1989.
37. Словарь по кибернетике / Под ред. В.С.Михалевича. - Киев: Гл. ред. УСЭ, 1989.
38. Советов Б. Я. Информационная технология. - М.: Высшая школа, 1992.
39. Становление информатики. Сб. статей из серии «Кибернетика». - М.: Наука, 1986.
40. Суханов А. П. Информация и прогресс. - Новосибирск: Наука, 1988.
41. Толковый словарь по искусственному интеллекту / Авторы-составители А-Н.Аверкин,

- М.Г.Гаазе-Рапопорт, Д.А.Поспелов. - М.: Радио и связь, 1992.
42. Уинстон П. Искусственный интеллект. - М.: Мир, 1980.
43. Урсул А.Д. Информатизация общества. Введение в социальную информатику. - М.: АОН, 1990.
44. Успенский В. А. Машина Поста. - М.: Наука, 1988
45. Успенский В. А., Семенов А. Л. Террия алгоритмов: основные открытия и приложения. - М.: Наука, 1987.
46. Хартли Р. Передача информации. В сб. «Теория информации и ее приложения». - М.: ИЛ, 1959.
47. Шеннон К.-Э. Работы по теории информации и кибернетике. - М.: ИЛ, 1963.

ГЛАВА 2

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

Многие вещи нам непонятны не потому, что наши понятия слабы, но потому, что сии вещи не входят в круг наших понятий

Козьма Прутков

ВВЕДЕНИЕ

Давно ушли в прошлое времена компьютеров первых поколений, когда аппаратные средства (часто называемые в нашей литературе жаргонным словом «железо» или американским термином «hardware») были главным предметом вожделения специалистов. В те времена, в 50-60-е годы, вопрос о программных средствах стоял так: если есть - хорошо, нет - сами напишем, только дайте ЭВМ. Сегодня, прежде всего в силу экономических обстоятельств, т.е. поменявшегося соотношения стоимости в диаде «аппаратные средства / программные средства» (последние часто зовутся «software»), и высокой стоимости разработки удовлетворяющих современным требованиям, но отсутствующих по той или иной причине, программных средств, при приобретении компьютера чаще обращают внимание на наличие для него доступного (по факту и по цене) программного обеспечения. На гигантских международных выставках типа СЕВИТ новинки программного обеспечения уже давно доминируют над новыми аппаратными разработками. Полный комплект программного обеспечения, необходимого для организации, скажем, автоматизированного рабочего места (АРМ) инженера-проектировщика, научного работника (физика, химика, биолога и т.д.) по стоимости превосходит (порой в несколько раз) стоимость компьютера адекватного класса.

Всевозможные программные средства, которых, видимо, насчитывается уже сотни тысяч для компьютеров различных типов, можно разделить на несколько классов в зависимости от назначения:

- операционные системы;
- системы программирования;
- инструментальные программные средства, интегрированные пакеты;
- прикладные программы.

Эти классы программных средств будут подробно рассмотрены в настоящей главе. В ней упоминаются десятки программных средств, о каждом из которых написано немало отдельных книг. О некоторых из этих программ ниже дано достаточно информации, чтобы начать работу с ними, но ни об одной нет исчерпывающих сведений. Для более полного знакомства с этими программными средствами нужно обратиться к встроенной помощи, справочникам, многочисленным книгам. Цель этой главы - сформировать понимание принципов работы с основными видами программного обеспечения персональных компьютеров, обработки информации с их помощью.

§ 1. ОПЕРАЦИОННЫЕ СИСТЕМЫ

1.1. НАЗНАЧЕНИЕ И ОСНОВНЫЕ ФУНКЦИИ ОПЕРАЦИОННЫХ СИСТЕМ

Особое место среди программных средств всех типов занимают операционные системы, являясь ядром программного обеспечения.

Операционная система - это комплекс программ, обеспечивающих

- управление ресурсами, т.е. согласованную работу всех аппаратных средств компьютера;
- управление процессами, т.е. выполнение программ, их взаимодействие с устройствами компьютера, с данными;
- пользовательский интерфейс, т.е. диалог пользователя с компьютером, выполнение определенных простых команд - операций по обработке информации.

Такое определение операционной системы уже апеллирует к ее функциям, поэтому рассмотрим эти функции подробнее.

Операционные системы — наиболее машиннозависимый вид программного обеспечения, ориентированный на конкретные модели компьютеров, поскольку они напрямую управляют их устройствами или, как еще говорят, обеспечивают интерфейс между пользователем и аппаратной частью компьютера.

В той мере, в какой это необходимо для понимания функций операционных систем, аппаратную часть компьютера можно представлять себе состоящей из следующих элементов:

- центрального процессора, имеющего определенную архитектуру (структуру регистров, набор и форму представления команд, формат обрабатываемых данных и т.д.) и характеризующегося производительностью, т.е. количеством простейших операций, выполняемых в единицу времени, а также другими качествами;
- оперативной памяти, характеризующейся емкостью (объемом) и скоростью обмена данными (прежде всего с центральным процессором);
- периферийных устройств, среди которых имеются
- устройства ввода (клавиатура, мышь, сканер и др.);
- устройства вывода (дисплей, принтер, графопостроитель и др.);
- внешние запоминающие устройства (дисководы для магнитных и оптических дисков, устройства для работы с лентами и др.);
- мультимедийные устройства.

Все эти аппаратные устройства обобщенно называют ресурсами компьютера.

В сравнении с оперативной памятью внешние запоминающие устройства обладают практически неограниченной емкостью. Так, емкость встроенного накопителя персональных компьютеров - винчестера - обычно в 50-100 раз больше объема оперативной памяти. Для других устройств - накопителей на гибких магнитных дисках и оптических дисках - используются сменные носители информации, однако время доступа к информации на внешних запоминающих устройствах значительно больше, чем к информации в оперативной памяти (в тысячи раз). Медленнее, чем центральный процессор, работают и устройства ввода - вывода.

За время существования компьютеров операционные системы претерпели значительную эволюцию. Так, первые операционные системы были однопользовательскими и однозадачными. Эффективность использования ресурсов компьютера в этом случае оказывалось невысокой из-за простоев всех, кроме одного работающего периферийного устройств компьютера. Например, при вводе данных простаивал центральный процессор, устройства вывода и внешние запоминающие устройства.

По мере роста возможностей, производительности и изменениях в соотношении стоимости устройств компьютера положение стало нетерпимым, что привело к появлению **многозадачных** операционных систем, остававшихся однопользовательскими.

Такие операционные системы обеспечивают постановку заданий в очередь на выполнение, параллельное выполнение заданий, разделение ресурсов компьютера между выполняющимися заданиями. Так, например, одно задание может выполнять ввод данных, другое - выполняться центральным процессором, третье - выводить данные, четвертое - стоять в очереди. Важнейшее техническое решение, обусловившее такие возможности, - появление у внешних устройств собственных процессоров (контроллеров).

При многозадачном режиме

- в оперативной памяти находится несколько заданий пользователей;
- время работы процессора разделяется между программами, находящимися в оперативной памяти и готовыми к обслуживанию процессором;

- параллельно с работой процессора происходит обмен информацией с различными внешними устройствами.

Наиболее совершенны и сложны **многопользовательские многозадачные** операционные системы, которые предусматривают одновременное выполнение многих заданий многих пользователей, обеспечивают **разделение ресурсов** компьютера в соответствии с приоритетами пользователей и **защиту данных** каждого пользователя от несанкционированного доступа. В этом случае операционная система работает в режиме **разделения времени**, т.е. обслуживает многих пользователей, работающих каждый со своего терминала.

Суть режима разделения времени состоит в следующем. Каждой программе, находящейся в оперативной памяти и готовой к исполнению, выделяется для исполнения фиксированный, задаваемый в соответствии с приоритетом пользователя интервал времени (интервал мультиплексирования). Если программа не выполнена до конца за этот интервал, ее исполнение принудительно прерывается, и программа переводится в конец очереди. Из начала очереди извлекается следующая программа, которая исполняется в течение соответствующего интервала мультиплексирования, затем поступает в конец очереди и т.д. в соответствии с циклическим алгоритмом. Если интервал мультиплексирования достаточно мал (~200 мс), а средняя длина очереди готовых к исполнению программ невелика (~10), то очередной квант времени выделяется программе каждые 2 с. В этих условиях ни один из пользователей практически не ощущает задержек, так как они сравнимы со временем реакции человека.

Одной из разновидностей режима разделения времени является фоновый режим, когда программа с более низким **приоритетом** работает на фоне программы с более высоким приоритетом. Работа в фоновом режиме реального времени аналогична работе секретаря руководителя. Секретарь занимается текущими делами до тех пор, пока начальник не дал срочное поручение.

Помимо рассмотренных режимов организации вычислительного процесса, все большее распространение получает схема, при которой ЭВМ управляет некоторым внешним процессом, обрабатывая данные и информацию, непосредственно поступающую от объекта управления. Поскольку определяющим фактором являются реально поступающие от объекта управления данные, такой режим называют **режимом реального времени**, а его организация возлагается на специализированную операционную систему.

Остановимся на некоторых понятиях, важных для понимания принципов функционирования всех операционных систем (ОС).

Понятие **процесса** играет ключевую роль и вводится применительно к каждой программе отдельного пользователя. Управление процессами (как целым, так и каждым в отдельности) - важнейшая функция ОС. При исполнении программ на центральном процессоре следует различать следующие характерные состояния (рис. 2.1):

- **порождение** - подготовку условий для исполнения процессором;
- **активное состояние** (или «Счет») - непосредственное исполнение процессором;
- **ожидание** - по причине занятости какого-либо требуемого ресурса;
- **готовность** - программа не исполняется, но все необходимые для исполнения программы ресурсы, кроме центрального процессора, предоставлены;
- **окончание** - нормальное или аварийное завершение исполнения программы, после которого процессор и другие ресурсы ей не предоставляются.

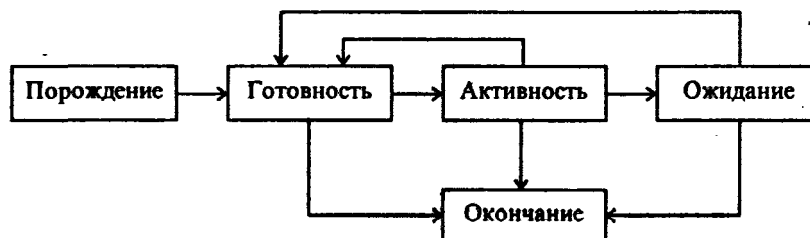


Рис. 2.1. Граф состояний переходов процесса из одной фазы в другую

Понятие «ресурс» применительно к вычислительной технике следует понимать как функциональный элемент вычислительной системы, который может быть выделен процессу на определенный промежуток времени. Наряду с физическими ресурсами -реальными устройствами ЭВМ - средствами современных операционных систем могут создаваться и использоваться виртуальные

(воображаемые) ресурсы, являющиеся моделями физических. По значимости виртуальные ресурсы - одна из важнейших концепций построения современных ОС. Виртуальный ресурс представляет собой модель некоего физического ресурса, создаваемую с помощью другого физического ресурса. Например, характерным представителем виртуального ресурса является оперативная память. Компьютеры, как правило, располагают ограниченной по объему оперативной памятью (физической). Функционально ее объем может быть увеличен путем частичной записи содержимого оперативной памяти на магнитный диск. Если этот процесс организован так, что пользователь воспринимает всю расширенную память как оперативную, то такая «оперативная» память называется виртуальной.

Наиболее законченным проявлением концепции виртуальности является понятие виртуальной машины, являющееся исходным при программировании на языках высокого уровня, например Паскале. Виртуальная машина есть идеализированная модель реальной машины, изолирующая пользователя от аппаратных особенностей конкретной ЭВМ, воспроизводящая архитектуру реальной машины, но обладающую улучшенными характеристиками:

- бесконечной по объему памяти с произвольно выбираемыми способами доступа к ее данным;
 - одним (или несколькими) процессами, описываемыми на удобном для пользователя языке программирования;
 - произвольным числом внешних устройств произвольной емкости и доступа.
- Концепция **прерываний** выполнения программ является базовой при построении любой операционной системы. Из всего многообразия причин прерываний необходимо выделить два вида: первого и второго рода. Системные причины прерываний первого рода возникают в том случае, когда у процесса, находящегося в активном состоянии, возникает потребность либо получить некоторый ресурс или отказаться от него, либо выполнить над ресурсом какие-либо действия. К этой группе относят и, так называемые, внутренние прерывания, связанные с работой процессора (например, арифметическое переполнение или исчезновение порядка в операциях с плавающей запятой). Системные причины прерывания второго рода обусловлены необходимостью проведения синхронизации между параллельными процессами.

При обработке каждого прерывания должна выполняться следующая последовательность действий:

- восприятие запроса на прерывание;
- запоминание состояния прерванного процесса, определяемое значением счетчика команд и других регистров процессора;
- передача управления прерывающей программе, для чего в счетчик команд заносится адрес, соответствующий данному типу прерывания;
- обработка прерывания;
- восстановление прерванного процесса.

В большинстве ЭВМ первые три этапа реализуются аппаратными средствами, а остальные - блоком программ обработки прерываний операционной системы.

В настоящее время используется много типов различных операционных систем для ЭВМ различных видов, однако в их структуре существуют общие принципы. В составе многих операционных систем можно выделить некоторую часть, которая является основой всей системы и называется ядром. В состав ядра входят наиболее часто используемые модули, такие как модуль управления системой прерываний, средства по распределению таких основных ресурсов, как оперативная память и процессор. Программы, входящие в состав ядра, при загрузке ОС помещаются в оперативную память, где они постоянно находятся и используются при функционировании ЭВМ. Такие программы называют резидентными. К резидентным относят также и программы-драйверы, управляющие работой периферийных устройств. Важной частью ОС является **командный процессор** - программа, отвечающая за интерпретацию и исполнение простейших команд, подаваемых пользователем, и его взаимодействие с ядром ОС. Кроме того, к операционной системе следует относить богатый набор **утилит** - обычно небольших программ, обслуживающих различные устройства компьютера (например, утилита форматирования магнитных дисков, утилита восстановления необдуманно удаленных файлов и т.д.).

1.2. ПОНЯТИЕ ФАЙЛОВОЙ СИСТЕМЫ

При наличии большого числа программ и данных необходим строгий их учет и систематизация. Операционным системам приходится работать с различными потоками данных, разными аппаратными и периферийными устройствами компьютера. Организовать упорядоченное управление всеми этими объектами позволяет файловая система.

На операционные системы персональных компьютеров наложила глубокий отпечаток концепция файловой системы, лежащей в основе операционной системы UNIX. В ОС UNIX подсистема ввода-вывода унифицирует способ доступа как к файлам, так и к периферийным устройствам. Под файлом при этом понимают набор данных на диске, терминале или каком-либо другом устройстве. Таким образом, файловая система - это система управления данными.

Файловые системы операционных систем создают для пользователей некоторое виртуальное представление внешних запоминающих устройств ЭВМ, позволяя работать с ними не на низком уровне команд управления физическими устройствами (например, обращаться к диску с учетом особенностей его адресации), а на высоком уровне наборов и структур данных. Файловая система скрывает от программистов картину реального расположения информации во внешней памяти, обеспечивает независимость программ от особенностей конкретной конфигурации ЭВМ, или, как еще говорят, логический уровень работы с файлами. Файловая система также обеспечивает стандартные реакции на ошибки, возникающие при обмене данными. Пользователь, работая в контексте определенного языка программирования, обычно использует файлы как поименованные совокупности данных, хранимые во внешней памяти и имеющие определенную структуру. При работе с файлами пользователю предоставляются средства для создания новых файлов, операции по считыванию и записи информации и т.д., не затрагивающие конкретные вопросы программирования работы канала по пересылке данных, по управлению внешними устройствами.

Наиболее распространенным видом файлов, внутренняя структура которых обеспечивается файловыми системами различных ОС, являются файлы с последовательной структурой. Такого рода файлы можно рассматривать как набор составных элементов, называемых логическими записями (или блоками), длина которых может быть как фиксированной, так и переменной, и доступ к которым - последовательный, т.е. для обработки (считывания или записи) i -й записи должна быть обработана предыдущая ($i-1$)-я запись.

В ряде файловых систем предусматривается использование более сложных логических структур файлов, чем последовательная. Например, записи в файле могут образовывать древовидные структуры, может использоваться индексно-последовательная организация файлов (с упорядочением записей по значению некоторых полей) или, так называемая, библиотечная структура файлов, использующая уровень учетной информации (каталога), облегчающей поиск и доступ к отдельным компонентам файлов. На физическом уровне блоки файла (обычно размером 256 или 512 байт) могут размещаться в памяти непрерывной областью или храниться несмежно. Первый способ хранения файлов, реализованный, например, в ОС РАФОС, приводит к затруднениям при изменении размеров файлов (т.е. к необходимости перезаписи файлов, если их длина увеличивается, или хранения «дыр», если длина уменьшается).

Наиболее развитый механизм несмежного распределения блоков файлов реализован в операционной системе UNIX, в которой размеры файлов могут динамически изменяться в пределах 1 Гбайта. Каждый файл в системе имеет дескриптор, в составе которого хранится список, содержащий 13 номеров блоков на диске и используемый для адресации к тем блокам, которые входят в состав файла. Первые десять элементов списка непосредственно указывают на десять блоков, в которых размещаются данные файла. В одиннадцатом элементе списка указан номер блока, хранящий список из 128 номеров блоков данных, которые принадлежат файлу (это первый уровень косвенной адресации). Двенадцатый элемент ссылается на блок, который содержит список из 128 номеров блоков первого уровня косвенной адресации (это второй уровень косвенной адресации). С помощью тринадцатого элемента указывается ссылка на блок, содержащий список из 128 номеров блоков второго уровня косвенной адресации.

Роль учетного механизма, позволяющего обслуживать десятки и сотни файлов, в файловой системе очень важна. Общим приемом является сведение учетной информации о расположении файлов на магнитном диске в одно место - его каталог (директорий). Каталог представляет собой список элементов, каждый из которых описывает характеристики конкретного файла, используемые для организации доступа к этому файлу - имя файла, его тип, местоположение на диске и

длину файла. В простых операционных системах (например ОС РАФОС) местоположение единственного каталога на магнитном диске (дискете) и его размер фиксированы. В более сложных системах каталог может находиться в любом месте диска, но на него должна иметься ссылка в, так называемой, метке тома, находящейся в фиксированном месте и формируемой при инициализации диска. Более того, каталогов может быть большое число и они могут быть логически связаны в какие-либо информационные структуры. Так, наиболее развитая многоуровневая файловая система UNIX поддерживает иерархическую (древовидную) систему каталогов (рис.2.2). Каждый пользователь может работать в составе этой структуры со своей системой каталогов (со своим поддеревом). Полное имя файла в данной структуре задает путь переходов между каталогами в логической структуре каталогов.

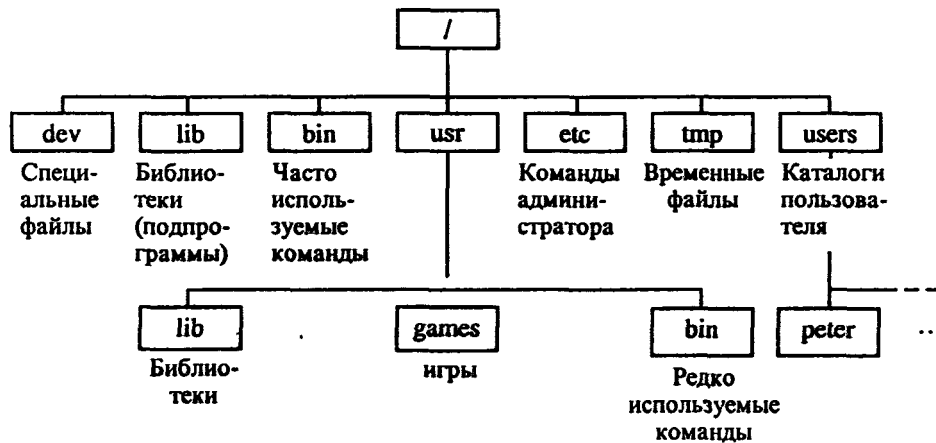


Рис. 2.2. Иерархическая система каталогов

Файл обладает уникальным идентификатором (именем), обеспечивающим доступ к файлу. Идентификатор включает в себя собственно имя - буквенно-цифровое обозначение файла, которое может содержать специальные символы (подчеркивание, дефис, ! и т.д.), и расширение имени файла (обычно отделяемое от имени файла точкой). Если имена создаваемых файлов пользователь может задавать произвольно, то в использовании расширений следует придерживаться традиции, согласно которой расширение указывает на тип файла, характер его содержимого. Например, в операционной системе MS-DOS файлы с расширениями

.com	
.exe	- исполняемые
.bat	
.txt	- текстовые
.doc	
.pas	- тексты программ на известных языках программирования: Паскаль, Бейсик,
.bas	Си, Фортран, соответственно
.c	
.for	
.dbf	- файл базы данных.

Известны десятки стандартных расширений, используемых при работе с различными программными системами.

В различных ОС существуют определенные ограничения на длину имени и расширения имени файла. Так, в MS-DOS длина имени файла не должна превышать восьми символов, а расширение - трех. В ОС UNIX ограничения значительно менее жесткие.

Имена директорией, начиная от корневого, образующие **путь к файлу**, отделяемые при записи друг от друга косой чертой (\ в DOS, / в UNIX), также как и обозначение диска, относятся к идентификатору файла. Например, в MS-DOS

d:\lang\pascal\work\example.pas

есть файл с именем example и расширением pas, указывающем на то, что это текст программы на Паскале, полный путь к которому:

d:- диск d;

\lang\pascal\work - это структура вложенных директорий, в самом внутреннем из которых находится необходимый файл example.pas.

Каждый каталог рассматривается как файл, имеет собственное имя. Продвижение по дереву при поиске некоторого каталога или файла возможно как вниз по дереву от текущего узла, так и вверх в направлении к корню. В каждом каталоге хранится список имен файлов, а также ссылки на дескрипторы файлов. В дескрипторах сосредоточена подробная информация о файле (список номеров блоков, занимаемых файлом, метод доступа к файлу, дата создания файла, идентификатор владельца, тип файла). В процессе работы могут создаваться новые каталоги и вписываться в требуемое место иерархии.

Файловая система ОС обеспечивает основные операции над файлами: их открытие (что сопровождается копированием учетной информации о файле, обеспечивающей быстрый доступ к нему, в некоторую структуру данных, расположенную в оперативной памяти, подготовкой буферов и каналов для передачи информации), копирование, перемещение, объединение, удаление, закрытие. Вторую группу представляют операции чтения и записи составных элементов файла. Особая группа операций обеспечивает печать содержимого каталогов или файлов, управление правами доступа к файлам, поиска файлов и т.д.

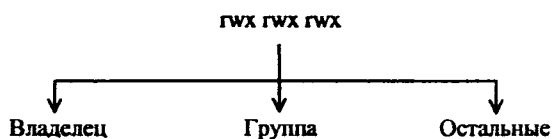
Развитые многопользовательские файловые системы обеспечивают также защиту и разделение данных, хранящихся в файлах, при работе с ними разных пользователей. Так, например, после входа в систему UNIX (который производится по паролю) пользователь получает доступ к ряду системных, групповых и личных каталогов и файлов. Каждый файл и каталог имеет владельца. Обычно это пользователь, создавший их. Владелец может затем назначить тип защиты файла от трех категорий пользователей:

- владельца (самого себя);
- представителей той же группы пользователей, что и владелец (понятие группы полезно при совместной работе над какими-либо проектами);
- всех остальных пользователей системы.

Каждый файл (каталог) имеет три вида разрешения на доступ:

- чтение (r - read) - можно просматривать содержимое файла (каталога);
- запись (w - write) - можно менять содержимое файла (создавать или удалять файлы в каталоге);
- выполнение (x - execute) - можно использовать файл как команду UNIX.

Комбинация видов доступа к файлу записывается последовательностью 9 символов:



Отсутствие права доступа обозначается минусом. Например: rwxr-x--x - файл может быть просмотрен, изменен и запущен на выполнение владельцем, просмотрен и выполнен членами группы, к которой относится владелец, и выполнен всеми остальными пользователями системы.

1.3. ОПЕРАЦИОННЫЕ СИСТЕМЫ ДЛЯ КОМПЬЮТЕРОВ ТИПА IBM PC

Общие сведения о MS DOS

Операционные системы для персональных ЭВМ за время существования этого класса компьютеров с 1975 г. претерпели значительное развитие, сопровождавшееся увеличением разрядности персональных компьютеров (ПК) от 8 до 32, расширением возможностей, улучшением интерфейса с пользователем (табл.2.1).

Таблица 2.1 Некоторые типы ОС для персональных компьютеров

ПК		
8-разрядные	16-разрядные	32-разрядные
P/M-80, MSX DOS, МикроDOS, Микрос-80	MS-DOS, РАФОС, ОС DBK, ИНМОС	UNIX, XENIX, Windows 95, OS/2

8-разрядные ОС сохраняют значение в качестве операционных систем простейших учебных и бытовых (игровых) компьютеров. Из-за ограниченного адресного пространства оперативной памяти (65 кбайт) серьезные профессиональные применения таких компьютеров невозможны.

16-разрядные IBM-совместимые компьютеры составляют значительную часть парка профессиональных персональных компьютеров в нашей стране. Самая распространенная ОС для этих компьютеров - однопользовательская однозадачная MS DOS (компания MicroSoft - сокращенно MS; DOS - английская аббревиатура названия «дисковая операционная система»). Первая версия этой ОС была создана одновременно с персональным компьютером IBM PC в 1981 г. и из внешних устройств поддерживала лишь накопители на гибких дисках с дискетами на 160 кбайт. Версия 2.0 связана с появлением модификации PC XT, поддерживала также накопители на жестких дисках до 10 Мбайт, древовидную файловую структуру. Популярная на протяжении ряда лет версия 3.3 (1987 г.) - для поддержки PC AT. Эта модификация ОС адресует 640 кбайт оперативной памяти, что в момент ее появления было прогрессивным моментом, а затем стало сдерживающим фактором прогресса программного обеспечения. Современные версии MS DOS преодолели ограничения на размер оперативного запоминающего устройства (ОЗУ), имеют множество новых команд, содержат встроенные драйверы устройств, графическую оболочку, справочную систему и т.д.

Основные структурные компоненты MS DOS таковы:

- базовая система ввода-вывода (BIOS);
- системный загрузчик (SB);
- драйверы устройств (т.е. программы, поддерживающие их работу);
- базовый модуль;
- командный процессор (называемый также интерпретатором команд);
- утилиты DOS (вспомогательные программы).

Охарактеризуем коротко основные компоненты. BIOS хранится в ПЗУ. Эта программа написана непосредственно в машинных кодах; при включении компьютера она автоматически считывается в ОЗУ, запускается на исполнение и производит беглую проверку работоспособности основных устройств компьютера. Затем BIOS производит поиск на дисках программы запуска операционной системы (программы **начальной загрузки**). BIOS имеет также функции поддержки стандартных периферийных устройств, прежде всего дисплея и клавиатуры.

Программа начальной загрузки, найденная BIOS-ом на диске, обращается последовательно к дисководам A, B и т.д. пока не найдет программу SB - **системный загрузчик**. Эта программа проверяет наличие на диске ядра операционной системы, состоящего из файлов с названиями `ibmio.sys` - файла расширения BIOS и `command.com` - командного процессора, загружает их в ОЗУ и запускает на исполнение первую из этих программ. Она дополнительно тестирует оборудование, осуществляет конфигурирование DOS (стандартное при отсутствии файла `config.sys` - файла конфигурации или нестандартное в соответствии с содержанием файла `config.sys`), подключает необходимые драйверы и т.д. Далее эта программа устанавливает некоторые указания о способах обработки прерываний (векторы прерываний) и передает управление базовому модулю DOS, который продолжает устанавливать правила обработки прерываний и после этого загружает в ОЗУ командный процессор и передает ему управление.

Пользователь, работающий с DOS без программ - оболочек или дополнительных интерфейсных систем, непосредственно общается с командным процессором. Режим работы - диалоговый, т.е. пользователь отдает команду, ОС выполняет и ждет следующей команды. Способ отдавать команды является достаточно архаичным - текст команды нужно просто набрать на клавиатуре, для чего большую часть команд надо помнить, а для редко встречающихся - пользоваться справочником (либо в виде книги, либо встроенным в DOS).

Командный процессор, будучи запущенным, вначале отыскивает и исполняет **программу автозапуска** (файл `autoexec.bat`), если она есть. Эта программа создается пользователем из команд

DOS для того, чтобы произвести некоторые рутинные действия по созданию удобной для начала работы обстановки. Например, если при запуске компьютера вы получаете на экране панели Norton Commander, то лишь потому, что «автозапуск» этой программы предусмотрен тем, кто составлял файл autoexec.bat. Следующее действие командного процессора - выдача на экран приглашения пользователю вводить команду, выглядящее, например, так: C> (если DOS загружалась с диска C).

В ходе работы прикладных программ в ОЗУ постоянно находится лишь малая часть DOS (называемая **резидентной**). Все остальные модули DOS подгружаются лишь по мере потребности в них и удаляются из ОЗУ после отработки.

Файловая система MS DOS поддерживает дисководы, обозначаемые латинской буквой и двоеточием, например:

a:, b:, c:,

иерархическую систему каталогов, заимствованную у системы UNIX, файлы с именами до восьми символов и расширением до трех.

Общие команды MS DOS

Общие команды распознаются и выполняются командным процессором command.com. Команды вводятся с клавиатуры, их ввод завершается нажатием клавиши <BBOД> (<ENTER>).

Общие команды DOS делятся на группы:

- команды работы с дисками;
- команды работы с файлами;
- команды работы с каталогами;
- команды управления системой.

Типовая структура команды выглядит следующим образом:

<имя команды> [<список параметров>] [<список ключей>]

Параметры (аргументы) указывают на те объекты, над которыми совершаются операции, ключи уточняют действие команды. Признак ключа (переключателя) -наличие косой линии ' /'. Квадратные скобки указывают на возможность отсутствия фрагмента.

DIR

Команда работы с каталогами; выводит на экран список директорией и файлов, находящихся внутри текущего директория. Если использовать команду DIR без параметров и переключателей, она выводит имена файлов (директорией), их расширения, размеры (в байтах), дату и время создания, их число, общий размер и размер свободного дискового пространства.

Полный синтаксис таков:

DIR [диск:] [путь] [имя_файла] [/ P] //W] [/A[:атрибуты]] [/O[:порядок_сортировки]] [...]

Параметры

[диск:] [путь] указывают дисковод и каталог, оглавление которого нужно просмотреть;
[имя_файла] указывают файл или группу файлов, список которых необходимо получить.

В имени файла могут быть использованы символы-заместители:

? заменяет один произвольный символ в имени файла;
* заменяет произвольное число произвольных символов.

Например:

DIR *.txt просмотр списка всех файлов с расширением txt;

DIR a?.* просмотр списка файлов с именами из двух знаков, первый из которых

буква а, и произвольными расширениями.

Ключи:

/P выводит информацию, пока экран не заполнится, для получения
 следующих экранов надо нажимать любую клавишу;
/W выводит информацию в сокращенном виде, только имена файлов и
 директориев (в 5 столбцов);
/A[:,] атрибуты] выводит информацию тех директориев и файлов, атрибуты
 которых указаны.

Вот некоторые атрибуты:

H . скрытые файлы;
- H все файлы, кроме скрытых;
S системные файлы;
- S все файлы, кроме системных;
D директории;
-D только файлы;
R файлы только для чтения.

Параметр

/O[:,] порядок_сортировки]

управляет порядком сортировки файлов в выдаваемом на экран списке. Без этого параметра имена файлов и директорией выдаются в алфавитном порядке. Задавая его соответствующим образом, можно организовать вывод файлов и директориев в порядке, обратном алфавитному, в алфавитном или обратном порядке по именам расширений, в порядке возрастания или убывания даты и времени последнего изменения содержимого файла или директория, в порядке возрастания или убывание их размеров.

Еще несколько команд той же группы (только имена):

MKDIR (MO) создание нового директория;
CHDIR (CD) переход в другой директорий.

DEL (ERASE)

Команда работы с файлами; удаляет файлы.

Синтаксис:

DEL [диск:] [путь] <имя_файла> [/P]

Параметр

[диск:] [путь] <имя_файла>

указывает местонахождение и имя удаляемого файла или группы файлов, если в имени используются символы-заместители.

Ключ /P вызывает запрос подтверждения для каждого удаляемого файла.

COPY

Команда работы с файлами; копирует один или более файлов в указанное место, а также может использоваться для слияния файлов. Синтаксис:

COPY [/Y|/Y] [/A|/B] <файл-источник> [/A|/B] [+ файл-источник [/A|/B] [+ ...]] [файл-результат [/A|/B]] [/V]

Параметры состоят из обозначения дискового, директория и имени файла.

<файл-источник> указывает местоположение и имя файла, содержимое которого необходимо копировать.

<файл-результат> указывает местоположение и имя файла, в который нужно поместить скопированную информацию.

Ключи:

/Y указывает, что команда не должна запрашивать подтверждения при замене существующих файлов;

/V проверка того, что новые файлы записаны правильно.

Еще команда той же группы:

RENAME (REN) - переименование файла или группы файлов;

Примерами команд управления системой служат (приводятся только имена):

COMMAND - запуск командного процессора;

EXIT - выход из командного процессора.

Дополнительные команды-утилиты

Помимо команд, распознаваемых и выполняемых командным процессором, в операционной системе имеется большое число утилит - команд, реализованных в виде отдельных программ. В качестве примера рассмотрим утилиту форматирования магнитных дисков.

FORMAT - форматирует диск для использования в MS DOS.

Утилита FORMAT создает пустой директорию и таблицы FAT на диске, а также проверяет наличие испорченных областей на диске. Может уничтожить все данные на диске.

Синтаксис:

FORMAT диск: [/V[:метка]] [/Q] [/U] [/F:размер] [/B/S] [/C]

FORMAT диск: [/V[:метка]] [/Q] [/U] [/T:дорожек\N:секторов] [/B/S] [/C]

FORMAT диск: [/V[:метка]] [/Q] [/U] [/I] [/4] [/B/S] [/C]

FORMAT диск: [/Q] [/U] [/1] [/4] [/8] [/B/S] [/C]

Параметр

диск: обозначает форматируемый диск (это единственный обязательный параметр утилиты).

Ключи

/V:метка указывает метку диска, используется редко;

/Q указывает, что производится «быстрое» форматирование, т.е. проверку испорченных областей проводить не надо;

/U указывает, что «восстанавливать» информацию до форматирования не потребуется;

/F:размер указывает емкость дискеты;

/S копирование на дискету файлов операционной системы IO.SYS, MSDOS.SYS и COMMAND.COM, что делает ее загрузочной;

/T:дорожек указывает число дорожек на дискете;

/N:секторов задает число секторов на дискете.

DISKCOPY

Команда работы с дисками (гибкими); копирует содержимое флоппи-диска в одном дисковом диске на диск в другом. Ее синтаксис таков

DISKCOPY [d1:][d2:][/I]

Здесь первые два объекта в квадратных скобках - параметры, третий - ключ.

Примеры.

DISKCOPY A: B: скопировать дискету в дисковом A на дискету в дисковом B;

DISKCOPY A: скопировать дискету в дисковом A на дискету в текущем дисковом;

DISKCOPY A: B: /I скопировать только первую сторону дискеты.

Еще несколько команд той же группы (только имена; параметры и ключи можно найти в справочниках):

DISKCOMP	- сравнение содержимого двух дискет (с целью определить, совпадает ли оно);
CHKDSK	- проверка целостности файловой структуры на диске, коррекция ее ошибок;
RECOVER	- восстановление (насколько возможно) информации на дефектном диске.

Большое количество утилит MS DOS описано в руководстве по этой системе. Важное значение имеют также драйверы, особенно расширенной оперативной памяти, входящие в состав ОС и позволяющие использовать более 640 кбайт памяти.

Особую роль в системе играют файлы CONFIG.SYS и AUTOEXEC.BAT, читаемые при загрузке системы и задающие ее конфигурацию, загружаемые в память драйверы и резидентные программы, а также дополнительные команды, выполняемые при загрузке системы.

CONFIG.SYS

Выполняется до загрузки командного процессора и содержит вызовы SYS-драйверов. Загружаемые драйверы устанавливаются командой DEVICE, после которой указывается полное имя файла, содержащего драйвер. Например, для подключения драйвера мыши MOUSE.SYS можно задать команду:

```
DEVICE=C:\DOS\MOUSE.SYS .
```

Начиная с версии MS DOS 4.0 предусматривается загрузка COM и EXE-драйверов с помощью команды INSTALL. Например,

```
INSTALL=C:\DOS\MOUSE.COM.
```

Для эффективной работы с различными типами микропроцессоров компьютера (80286, 80386, 80486, Pentium) и размеров оперативной памяти используют специальные драйверы:

```
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=C:\DOS\EMM386.EXE NOEMS
DEVICE=C:\DOS\EMM486.EXE.
```

Кроме загрузки внешних драйверов, CONFIG.SYS загружает свои (внутренние) команды.

Если на компьютере отсутствует кэш жесткого диска (т.е. буферная область ОЗУ, где сохраняется содержание блоков диска), то для ускорения работы с диском вводят команду BUFFERS. Буфер - это часть оперативной памяти размером 532 байт.

Пример:

```
BUFFERS=20.
```

С помощью команды FILES можно указать число файлов, которые могут быть одновременно использованы системой и программами.

Команда DOS дает возможность загружать модули операционной системы и некоторые драйверы в область памяти выше 640 кбайт, тем самым увеличивая размер свободной базовой памяти, что важно для ряда прикладных программ.

Ниже приведены примеры типичных файлов конфигураций:

1. для PC 286

```

REM Типичный CONFIG.SYS
DOS=HIGH
REM По возможности загружать модули операционной системы и
REM некоторые драйверы в HMA
REM (первые 64 кбайта области памяти выше 1 Мбайта)
FILES=20
REM До 20 файлов может быть одновременно открыто
BUFFERS=5
REM Для работы с файлами на жестком диске использовать 5 буферов
DEVICE=C:\DOS\HIMEM.SYS
REM Загрузка менеджера области памяти выше 1 Мбайта
DEVICE=C:\WINDOWS\MOUSE.SYS
REM Загрузка драйвера манипулятора типа «мышь»
DEVICE=C:\STACKER\STACHIGH.SYS
REM Загрузка драйвера поддержки работы с дисками,
REM использующими компрессию данных

```

2. для PC 386

```

REM Типичный CONFIG.SYS
DOS=HIGH, UMB
REM По возможности загружать модули операционной системы и
REM некоторые драйверы в HMA
REM (первые 64 Кбайта области памяти выше 1 Мбайта)
REM и UMB (блоки в области памяти между 640 Кб и 1 Мб)
FILES=20
REM До 20 файлов может быть одновременно открыто
BUFFERS=5
REM Для работы с файлами на жестком диске использовать 5 буферов
DEVICE=C:\DOS\HIMEM.SYS
REM Загрузка менеджера области памяти выше 1 Мбайта
DEVICE=C:\DOS\EMM386.EXE NOEMS
REM Загрузка менеджера расширенной памяти
REM с включенной поддержкой работы с UMB
DEVICEHIGH=C:\WINDOWS\MOUSE.SYS
REM Загрузка драйвера манипулятора типа «мышь»
REM в UMB
DEVICE=C:\STACKER\STACHIGH.SYS
REM Загрузка драйвера поддержки работы с дисками,
REM использующими компрессию данных

```

Не менее важную роль при начальной загрузке компьютера играет файл AUTOEXEC.BAT. Фактически в AUTOEXEC.BAT записаны команды, которые иначе пришлось бы вводить вручную в начале каждого сеанса работы. Например:

```

C:\KEYRUS
REM запуск программы, руссифицирующей ввод с клавиатуры и вывод на экран
C:\DOS\MOUSE
REM запуск драйвера манипулятора типа «мышь»
PROMPT $P$G
REM установка вида приглашения командной строки DOS
PATH C:\:C:\DOS:C:\NC:C:\TOOLS
REM установка путей поиска файлов программ, вызываемых на выполнение
SET TEMP=C:\DOS
REM установка значения переменной окружения, указывающей путь
REM к временным файлам
NC
REM запуск файл-менеджера NORTON COMMANDER

```

1.4. ОБОЛОЧКИ ОПЕРАЦИОННЫХ СИСТЕМ

Интерфейс операционной системы DOS не обладает необходимой дружелюбностью. Ко-

манды нужно знать наизусть, посимвольно набирать на клавиатуре и при этом не допускать ошибок. Все это предъявляет высокие требования к квалификации пользователя.

Для облегчения взаимодействия пользователя с компьютером существуют, так называемые, оболочки операционных систем - программы, делающие наглядным и простым выполнение базовых операций над файлами, каталогами и др. с использованием меню, защитой от необдуманных и ошибочных действий и разветвленной контекстной помощью.

Простая оболочка обычно входит в комплект утилит операционной системы MS DOS. Однако, значительно большее распространение получила оболочка под названием «нортон командер» (Norton Commander). Остановимся на ее интерфейсе подробнее, рис. 2.3.

При работе Norton Commander в стандартной настройке (конфигурации) на экране дисплея имеются следующие области:

- правая и левая панель - большие синие прямоугольники, на которых отображаются каталоги (директории) дисков; одна из панелей является активной; обозначение текущего диска и директория вверху активной панели выделены цветом; внутри панели находится указатель;
- командная строка с приглашением, в которой можно непосредственно набирать команды DOS или формировать их с помощью имен файлов на панелях;
- строка подсказки с обозначением команд, закрепленных за функциональными клавишами F1... F10.

Указатель перемещается внутри панели при нажатии клавиш управления курсором; переключить активную панель можно с помощью клавиши <Tab>. Чтобы войти в директорию, нужно указать его имя указателем и нажать клавишу <Ввод> (<Enter>), чтобы выйти из текущего директория - установить указатель на две точки вверху панели выше имен файлов и директориев и нажать клавишу <Ввод> (<Enter>). При нажатии клавиши <Ввод>, когда указатель установлен на имени исполняемого файла (с расширением .bat или .exe или .com), происходит запуск этого файла на исполнение. Если необходимо вызвать имя файла в командную строку для формирования параметров команды, нужно, выделив имя файла указателем, нажать одновременно <Ctrl> и <Enter>. Можно по желанию определить действия Norton Commander над файлами с произвольными расширениями при нажатии клавиши <Ввод>. Так, можно задать, что при указании файла с расширением .txt запускается текстовый редактор и в него загружается указанный файл.

C:\AIDS							
Name	Size	Date	Time	Name	Size	Date	Time
..	»UP--DIR«	7.03.99	11:30	ACROBAT3	»SUB-DIR«	27.02.99	13:55
AIDSTEST	»SUB-DIR«	7.03.99	11:30	ASUS	»SUB-DIR«	27.02.99	13:34
DRWEB	»SUB-DIR«	7.03.99	11:30	BOOKS	»SUB-DIR«	27.02.99	13:56
PRISTAV	»SUB-DIR«	7.03.99	11:30	CONTEXT3	»SUB-DIR«	27.02.99	13:24
				COREL	»SUB-DIR«	27.02.99	13:24
				DISTR3	»SUB-DIR«	27.02.99	13:57
				DTEXT23	»SUB-DIR«	27.02.99	13:27
				EMTEX	»SUB-DIR«	27.02.99	13:27
				IDL	»SUB-DIR«	27.02.99	13:27
				HOW	»SUB-DIR«	27.02.99	14:05
				NPL	»SUB-DIR«	27.02.99	13:30
				PN65	»SUB-DIR«	27.02.99	14:05
				PSFONTS	»SUB-DIR«	27.02.99	14:05
				RECYCLED	»SUB-DIR«	27.02.99	13:37
				RSI	»SUB-DIR«	27.02.99	13:32
				SB16	»SUB-DIR«	27.02.99	13:33
				SBOD	»SUB-DIR«	27.02.99	13:33
..	»UP--DIR«	7.03.99	11:30	PKFONTS	»SUB-DIR«	27.02.99	13:31

Рис. 2.3. Типичный вид панелей Norton Commander

Основные операции выполняются при нажатии функциональных клавиш. Так, при нажатии клавиши

F1 происходит вызов контекстной подсказки (гипертекста с развитой системой ссылок);

F2 вызывается пользовательское меню, в котором по желанию собраны часто выполняемые команды;

F3 вызывается для просмотра (View) в режиме текста или кодов файл, указанный на активной панели;

F4 вызывается простейший редактор для редактирования указанного на активной панели текстового файла;

F5 копируется указанный файл или директорий (или группа отмеченных файлов и директорий) с активной панели на диск, отражаемый пассивной панелью;

F6 происходит переименование (если вводится новое имя) или перемещение файлов или директорий с активной панели на пассивную;

F7 создается директорий на активной панели;

F8 удаляются указанные или отмеченные файлы и директории;

F9 вызывается горизонтальное меню в верхней строке экрана;

F10 происходит выход из Norton Commander.

Выделение файлов происходит при нажатии клавиши <Ins> или при задании фильтра с помощью серой клавиши «+».

Горизонтальное меню, вызываемое клавишей F9, позволяет изменить настройки Norton Commander (например, режим отображения информации на панелях), выполнить перечисленные и некоторые дополнительные, важные для пользователя, команды и действия.

Примечательно, что большинство операций можно выполнить с помощью Norton Commander несколькими способами:

- с помощью меню, выбирая команду с помощью клавиш управления курсором;
- с помощью меню, выбирая команду с помощью манипулятора «мышь»;
- с помощью меню, выбирая команду с помощью выделенной в команде буквы;
- с помощью «горячих клавиш» - сочетания клавиш при их одновременном нажатии.

По мере роста квалификации пользователи переходят к использованию именно «горячих клавиш». Например, чтобы перейти к другому дисководу на левой панели, нажимают комбинацию <Alt>+<F1>, на правой - <Alt>+<F2>; чтобы отключить панели - <Ctrl>+<0> и т.д.

Norton Commander имеет-резидентную часть, которая восстанавливает состояние Norton Commander после окончания работы прикладных программ.

Наряду с Norton Commander, нашли применение аналогичные программные средства типа Volkov Commander и DOS Navigator. Имея много общего с Norton Commander, они оказываются более удобными в ряде специфических моментов (таких как работа с архивами, подключение внешних редакторов, соединение файлов и т.д.).

С переходом на персональные компьютеры с процессором 80386 и с увеличенной памятью (не менее 4 Мбайт), на смену Norton Commander и характерному для DOS стилю работы в текстовом режиме пришла оболочка Windows и новый стиль работы с графическим интерфейсом. Идея графического интерфейса Windows заимствована компанией «Microsoft», долгое время специализировавшейся на операционных системах для персональных компьютеров IBM, у операционной системы для компьютеров Apple. Иногда Windows определяют не как оболочку, а как нечто большее, используя термин типа «операционная среда». При этом исходят из того, что если классическая оболочка (такая как Norton Commander) видоизменяет лишь пользовательский интерфейс, то программа типа Windows дополнительно к этому берет на себя управление программами и заданиями, т.е. реализует основные функции операционной системы.

Принципиально важные особенности Windows по сравнению с MS DOS - это многозадачность (допускается одновременное выполнение нескольких процессов) и возможность обмена данными между работающими программами. Важно и то, что Windows использует расширенную оперативную память (много больше 640 кбайт) и подразумевает единый интерфейс всех прикладных программ. Недаром утвердилось понятие «программировать под Windows», т.е. ориентироваться на стандартный графический интерфейс.

Название Windows - «окна» - говорит само за себя. Эта оболочка операционной системы построена на основе графических окон, соответствующих программным средствам и группам программных средств, которыми пользователь может управлять, изменять их размеры, перемещать по экрану, открывать и закрывать по своему желанию.

Оболочка Windows ориентирована на работу с помощью манипулятора «мышь». Все операции в этой среде в высокой степени унифицированы, все программные средства имеют очень схожие интерфейсы и принципы управления, что значительно ускоряет освоение новых программных средств,

Первой версией обсуждаемой операционной среды, получившей широкое распространение, стала Windows 3.1 (Windows for work groups 3.11), рис. 2.4.

При запуске Windows 3.1 пользователь видит на экране окно Program Manager -Диспетчера

Программ, в котором имеются пиктограммы (условные схематические обозначения) программ; среди них всегда имеются группы аксессуаров (accessories) и приложений (main), а также другие группы, такие как Microsoft Office. Группу можно активизировать, указав ее пиктограмму курсором и дважды нажав левую кнопку мыши; при этом открывается окно, содержащее пиктограммы для каждого программного средства, относящегося к группе. Чтобы запустить программное средство, надо указать его курсором и дважды нажать левую клавишу мыши.

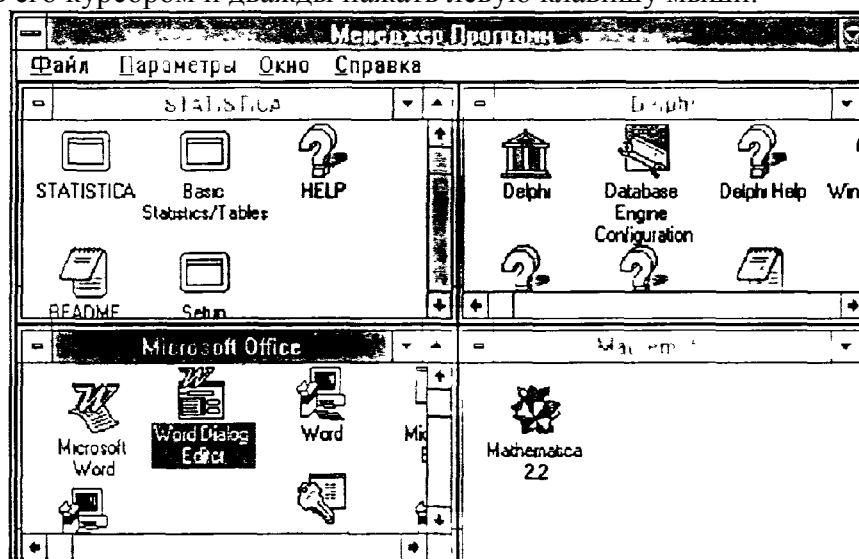


Рис.2.4. Типичный вид среды Windows 3.1

Работающую программу можно снять, указав знак «<-» в левом верхнем углу окна и нажав левую кнопку мыши, или отложить (минимизировать), указав знак «<-» в правом верхнем углу. Может быть минимизирована и группа, если нажать левую клавишу мыши, предварительно указав на «<-» в правом верхнем углу окна группы.

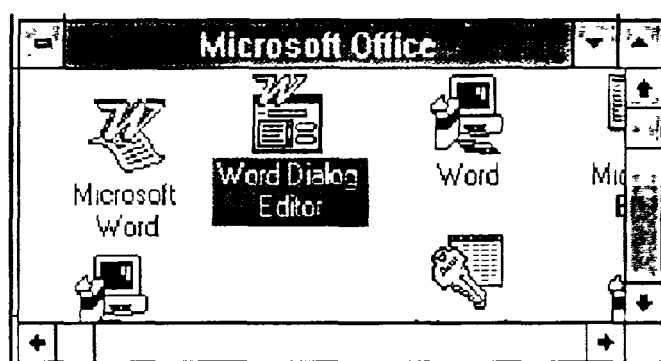


Рис. 2.5. Пример группового окна

Существует группа программ, составляющих стандартные приложения Windows. Это относительно небольшие по возможностям (по сравнению со специализированными) прикладные программы: текстовый редактор Write, графический редактор PaintBrush, *картотека*, *календарь*, *калькулятор*, *часы* и т.д. Конечно, возможности редактора Write при оформлении книги не идут в сравнение с текстовым процессором Word-7, но она существенно проще в освоении. В *Картотеке* можно хранить «карточки» с текстами и рисунками и вести поиск информации (например, по ключевым словам), т.е. организовать простую базу данных. *Часы*, *Календарь* и *Калькулятор* удобно всегда иметь под рукой.

Совместная работа нескольких программ требует возможности обмена данными между ними. При разработке программного обеспечения для MS DOS об этом особо не заботились: даже если разные программы могут обрабатывать один и тот же файл (например, редакторы Лексикон и MultiEdit могут поочередно искать орфографические ошибки в смешанном русско-английском тексте), то надо вначале выйти из одной программы и затем войти в другую, имеющую, как правило, иначе организованный интерфейс, и т.д. В Windows такой проблемы нет: можно, например, ввести в текст, создаваемый с помощью Write или Word, рисунок, созданный с помощью Paint-

Brush, не выходя ни из одной из этих программ в едином интерфейсе Windows.

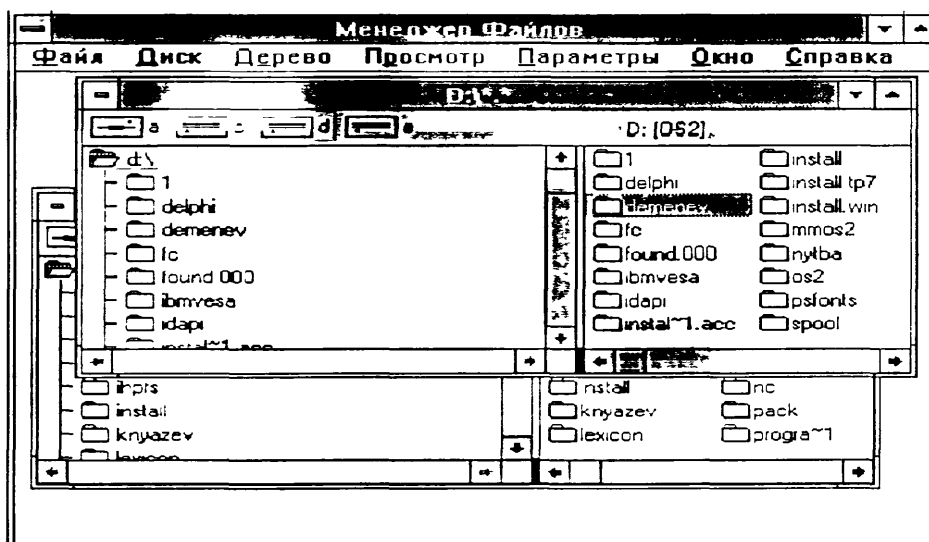


Рис. 2.6. Пример окна диспетчера файлов

Для работы с файлами в Windows существует специальная программа - File Manager (Диспетчер Файлов - Менеджер Файлов), рис. 2.6. Она позволяет выполнять все виды работ с файлами: просмотр файловой структуры, копирование, удаление, переименование, пересылку между каталогами и т.д. Пиктограмма Диспетчера Файлов имеет вид шкафа с ящиками, каталога - папки, текущего каталога - раскрытой папки. Благодаря наглядности и простоте использования оболочка Windows получила огромное распространение и стала стандартом для IBM совместимых персональных компьютеров с процессорами 386 и более мощных.

Важно понимать, однако, что многозадачность Windows не есть реальная параллельность в выполнении программ. На машине с одним процессором, которой является IBM PC, истинная параллельность невозможна. Среда Windows создает видимую параллельность, разделяя время между приложениями малыми порциями и постоянно переходя от одного к другому.

С 1995 года на смену операционной оболочке Windows 3.1 для IBM-совместимых персональных компьютеров пришла операционная система Windows'95. В настоящее время эта операционная система бьет все рекорды популярности. По прогнозам ожидается около 100 млн. продаж этой системы к 2000 году. Трудно оценить количество несанкционированных копий этой системы, широко используемой как в России, так и во всем мире. Windows'95 предназначена для установки на персональных компьютерах desktop и компьютерах типа notebook, имеющих процессор типа

Intel 80386DX, 80486, Pentium. Объем оперативной памяти должен быть не менее 4 Мбайт. Новая версия операционной системы лучше использует дополнительную оперативную память, чем предыдущая. Однако многие приложения для нормальной работы требуют от 8 до 16 Мбайт.

Новая версия обеспечивает более высокую производительность и большие возможности за счет применения 32-, а не 16-разрядной передачи данных. Windows'95 является высокопроизводительной, универсальной, надежной, многозадачной и многопоточковой интегрированной 32-разрядной операционной системой нового поколения с расширенными сетевыми возможностями, работающей в защищенном режиме и обеспечивающей графический интерфейс с пользователем. Windows'95 представляет собой интегрированную среду, обеспечивающую эффективный обмен информацией между отдельными программами и предоставляющую пользователю широкие возможности по обработке текстовой, графической, звуковой и видеoinформации. Понятие интегрированности подразумевает также совместное использование ресурсов компьютера всеми программами.

Операционная система позволяет прослушивать компакт-диски, редактировать музыкальные произведения, работать с видеофрагментами. Новая операционная система обладает также улучшенными телекоммуникационными возможностями, которые позволяют использовать ее в локальных и глобальных сетях, работать с электронной почтой. Windows'95 обеспечивает работу пользователя в сети, с электронной почтой, факсом и со средствами мультимедиа, поддерживает большинство приложений DOS и предыдущих версий Windows. Если в предыдущих версиях ак-

тивное приложение периодически передавало системные ресурсы другим приложениям, работающим в фоновом режиме, то новая операционная система сама управляет ресурсами, используя принцип вытесняющей многозадачности. Приложение, нуждающееся в ресурсах, может приостановить работу до получения ресурса или перейти к другим операциям. Многопоточное выполнение одной задачи позволяет при задержке в выполнении одного потока работать со следующим. Распределение времени между потоками производится с учетом их приоритетов. Приложения работают в защищенном адресном пространстве. После завершения работы приложения память автоматически очищается.

Применение 32-разрядного фонового спулинга печати ускоряет печать больших документов, позволяет минимизировать перерывы в работе. Сопоставление возможностей Windows'95 и Windows for Workgroups 3.11 показало, что при использовании новой операционной системы скорость загрузки и печати 100-страничного WinWord-документа возросла примерно на 30%.

После загрузки Windows'95 на экране появляется изображение, напоминающее рабочий стол. Так же, как на рабочем столе, на его модели (на экране) размещены значки папок с документами и значки быстрого доступа. Одна из основных задач, поставленная разработчиками новой операционной системы, заключалась в обеспечении простоты, удобства, интуитивной очевидности пользовательского интерфейса. Интерфейс Windows'95 спроектирован так, чтобы создать комфортные условия для пользователя и обеспечить объектно-ориентированную и документированную работу. По сравнению с предыдущими версиями улучшилось оформление экрана: появились эффектные трехмерные интерфейсные элементы, изменилось оформление диалогового окна, окон приложений и документов. Улучшенный пользовательский интерфейс, однотипность выполнения всех базовых операций призваны ускорить процесс освоения операционной системы.

Для обозначения различных объектов, с которыми имеет дело пользователь, в Windows используются графические символы. В Windows'95 отдельные файлы объединяют в папки, которые являются аналогом каталогов, использовавшихся в предыдущих версиях. Так же, как в каталоге может находиться несколько каталогов более низкого уровня, папка может состоять из нескольких папок более низкого уровня.

Значительное внимание уделено документо-ориентированной работе с тем, чтобы пользователь в первую очередь уделял внимание документам, а не прикладным программам (документом называется любой файл, обрабатываемый с помощью прикладной программы). Windows'95 позволяет открыть любой документ, не запуская предварительно приложение, в котором оно создано. Объекты (предметы), с которыми мы контактируем в реальной жизни, обладают определенными свойствами. У каждого предмета свой внешний вид, вес, габариты и т.п. Аналогично, объекты Windows имеют свои характеристики. Можно подобрать внешний вид значка, отображающего файл. Файлы имеют размеры, для них задаются атрибуты и т. п.

Windows'95 предоставляет удобные средства быстрого вызова программ, документов и папок с помощью значков быстрого вызова, позволяющих двойным щелчком открыть папку или документ, не запуская предварительно приложение, в котором создавался объект. Чтобы ускорить открытие часто используемых документов и запуск приложений, можно создать к ним сколько угодно значков быстрого вызова и разместить их на рабочем столе в одной или нескольких папках. Например, можно обеспечить быстрый вызов принтера, установив его значок на рабочей поверхности стола. В этом случае, чтобы распечатать файл будет достаточно перетащить мышью его значок на значок принтера. Двойной щелчок значка быстрого вызова *Блокнот* на экране дисплея запустит текстовый процессор.

Чтобы установить значок быстрого вызова к папке (файлу), ее необходимо выделить в окне *Мой компьютер* и выбрать команду *Создание ярлыка* из меню *Файл*. Первоначально значок располагается в конце списка окна. Значок можно переместить или скопировать на рабочий стол или в часто используемую папку с помощью мыши методом Drag and Drop (Перетащить и Отпустить). Другой вариант установки значка быстрого вызова в нужной папке или на рабочем столе - перетащить значок программы или документа в нужную папку, нажав правую кнопку мыши, и воспользоваться командой *Создать ярлык* из динамического меню, которое появится, когда отпустят правую кнопку. Можно не копировать файл (папку) в другую папку, а вставить в нее значок быстрого вызова к этому файлу. Сначала следует выделить файл, затем активизировать команду *Копировать* из меню *Правка*. После перехода в окно, где предполагается разместить значок быстрого вызова, активизируют команду *Вставить ярлык* из меню *Правка*. Изображение значка быстрого

.вызова можно изменить с помощью диалогового окна, появляющегося после выделения значка и активизации команды *Свойства* из меню *Файл*. Все значки быстрого вызова связаны с файлами и папками, которые они представляют. При удалении файлов и папок автоматически удаляются и значки. При удалении значка быстрого вызова файл, с которым он связан, не удаляется. Если переименовать папку или файл, надпись к значку не меняется, однако, связь между ними сохраняется.

Длина имени файла в предыдущих версиях Windows, как и в DOS, была ограничена восемью символами, а расширение - тремя символами после точки. Это вызывало неудобства при вспоминании содержания файла. В Windows'95 эти ограничения сняты. Windows'95 позволяет давать файлам имена, содержащие до 255 символов и включать пробелы, знак плюс, знак равенства, квадратные скобки, точку с запятой и другие знаки препинания. Пробелы, находящиеся в начале и в конце имени, не учитываются. Имя файла можно писать на русском языке. Любые символы, стоящие после последней точки, рассматриваются как расширение, расширение имени зависит от приложения, в котором создавался файл. Имя для папки задается так же, как для файла. Однако для папки не задается расширение.

Чтобы установить атрибуты файла и определить его принадлежность к приложению или документу, используются первые три символа после последней точки в имени файла. Так, для файла CONFIG.SYS устанавливается расширение SYS. Совместимость имен файлов новой операционной системы с предыдущими версиями и DOS обеспечивается поддержкой ранее используемой таблицы размещения файлов (FAT), в которой хранится информация о свободных секторах и о дисковом пространстве, отведенном для файлов. Существенные ограничения таблицы FAT связаны с тем, что она не предназначена для работы с большими дисками объемом ≥ 100 Мбайт. При работе с такими дисками FAT не помещается целиком в памяти и загружается частями, что увеличивает количество перемещений головок при считывании файла. Кроме того, использование FAT приводит к сильной фрагментации больших дисков, что увеличивает потери времени, связанные с их обработкой. Аналогичная таблица для Windows'95 совместима с FAT и поэтому при установке Windows'95 не требуется переформатирование жесткого диска. При использовании длинного имени, созданного Windows'95, в DOS FAT создаются «псевдонимы», которые обеспечивают ввод первых восьми букв из нового имени и добавляют порядковый номер после знака тильды "~". Например, при использовании слова literature (литература) в именах двух файлов в DOS они запишутся как litera~1 для первого документа и litera~2 для второго документа. Инсталлируемая файловая система Windows'95 поддерживает также сетевые файловые системы.

Существенно усовершенствованы в Windows'95 окна. Групповые окна заменены на окна папок, в которых появились очень полезные для работы панели инструментов. Их можно отобразить в окне папки или удалить командой *Панель инструментов* (рис. 2.7.) из меню *Вид*. Панель содержит раскрывающееся окно списка, в котором представлено имя текущей папки, и кнопки, дублирующие часто используемые команды. После раскрытия окна списка в нем видна древовидная диаграмма папок, имеющихся на компьютере. Команду можно быстро активизировать, щелкнув кнопку на панели инструментов. При этом нет необходимости сначала выбирать меню. При подводе указателя мыши к кнопке рядом с ним появляется флажок-подсказка с названием выполняемой команды.

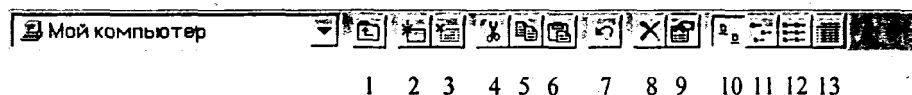


Рис. 2.7. Панель инструментов окна папки

Ниже кратко описывается назначение кнопок:

- 1 - вверх на один уровень (Up one level) - переход в родительскую папку;
- 2- подключить сетевой диск (доступно при работе в сети);
- 3 - отключить сетевой диск (доступно при работе в сети);
- 4 - вырезать (Cut) - перемещение выделенного объекта в Буфер Обмена;
- 5 - копировать (Copy) - копирование выделенного объекта в Буфер Обмена;
- 6 - вставить (Paste) - вставка выделенного объекта из Буфера Обмена;
- 7 - отменить (Undo) - отмена последней команды, восстанавливает вид окна до выполнения

предыдущей команды;

8 - удалить (Delete) - уничтожение выделенного файла или папки;

9 - свойства (Properties) - вызов одноименного диалогового окна;

10 - большие значки (Large Icons) - отображение рядом с именами папок и файлов больших значков;

11 - маленькие значки (Small Icons) - отображение рядом малых значков;

12 - список (List) - отображение сначала папок, затем файлов столбцами так, чтобы надписи к ним следовали в алфавитном порядке;

13 - таблица (Details) - отображение рядом с именами папок и/или файлов их кратких характеристик.

В зависимости от указанного типа файла, Windows'95 позволяет открыть то или иное приложение, использовать определенный набор команд. Для изменения типа файла или значка к нему используют кнопку *Правка*.

При работе с документом следует максимально увеличить область экрана, где может располагаться интересующая нас информация. Однако большую часть экрана часто занимают различные панели. Отображение на экране панели задач регулируется с помощью флажков вкладки *Параметры панели задач* команды *Панель задач* меню *Настройки*. Установка флажка *Автоматически убирать с экрана* позволяет не показывать панель задач и освободить максимум рабочего пространства для работающего приложения.

Windows'95 дает возможность печатать документы на нескольких десятках языков. При начальной установке системы необходимо записать утилиты, позволяющие работать на нужном вам языке. Русифицированная версия содержит кодовую страницу, поддерживающую знакогенератор и раскладку клавиатуры на русском языке. Эта страница обеспечивает корректную работу с именами файлов, содержащими русские буквы не только в Windows, но и в DOS. Язык, на котором будут печататься символы, переключатели, используемые для перехода с одного языка на другой, выбираются в диалоговом окне программы *Клавиатура* с вкладкой *Язык* приложения *Панель управления*. Приложение можно вызвать, щелкнув кнопку *Пуск*. Затем в меню *Настройки* выбирают команду *Панель управления*. Чтобы добавить другой язык для ввода символов, нажимают кнопку *Добавить*, в раскрывающемся окне списка *Язык* выбирают требуемый и щелкают кнопку *ОК*.

Для выделения заголовков, смыслового разграничения отдельных фрагментов, при написании формул, индексов используются различные стили и размеры шрифта. Получить справочные данные о шрифте и увидеть его гарнитуру (стиль) позволяет окно, появляющееся после выбора команды *Открыть* из меню *Файл* папки *Шрифты* программы *Мой компьютер*. В окне отражаются название, размер файла, версия, фирма-разработчик, демонстрируются образцы нескольких размеров шрифта. Чтобы удалить используемый шрифт, его надо выделить в окне папки *Шрифты* и выбрать команду *Удалить* из меню *Файл*.

Во многих приложениях гарнитура и размер шрифта задаются в диалоговом окне команды *Шрифт*. Перечень всех установленных шрифтов приводится в окне списка *Шрифт*. Размер в пунктах выбирается в окне *Размер*. Образец написания выбранного шрифта представляется в демонстрационном поле *Образец* в правой части окна. В поле *Цвет* задается цвет символов. Начертание (стиль) шрифта задается в окне *Начертание*: обычный, полужирный, курсив, подчеркнутый.

Меню *Пуск* позволяет выполнить большой набор работ, связанных с запуском приложений, получением справок, поиском и открытием документов, настройкой системы. Видимо поэтому разработчики операционной системы рядом с кнопкой *Пуск* поместили подсказку: «Начните работу с нажатия этой кнопки». Для активизации меню *Пуск* следует щелкнуть кнопку *Пуск* или нажать на клавиши *Ctrl+Esc*. При остановке указателя мыши на пункте меню со значком треугольника с правой стороны раскрываются окна, содержащие подменю и команды. В свою очередь отдельные пункты появившегося подменю также могут быть отмечены значком треугольника и иметь свои подменю. Каждое подменю содержит группу программ. Для выбора программы необходимо остановить на ней указатель и щелкнуть мышью.

Пункт *Документы* открывает список с названиями последних документов, с которыми работал пользователь. Список может содержать до 15 наименований документов, независимо от приложения, в котором они создавались. Для открытия документа следует щелкнуть на его названии. Следует отметить, что некоторые приложения не добавляют имена файлов в список меню

Документы. Тогда документ можно открыть, запустив то приложение, в котором он создавался. Как правило, открыть документ позволяет команда *Открыть* из меню *Файл* соответствующего приложения Windows. Кроме того, документ можно открыть двойным щелчком его значка в окне *Мой Компьютер*.

Ряд приложений Windows проверяет, имеются ли в памяти компьютера несохраненные данные. При выходе из приложения без команды *Сохранить* появится предупреждающее сообщение с вопросом о необходимости сохранения последних изменений. При попытке закрыть приложение без указаний как поступить с открытым документом появится запрос: «Сохранить изменения, внесенные в документ?». Три кнопки *-Да, Нет, Отменить* - позволяют сохранить внесенные во время текущего сеанса работы изменения, не вносить изменений или отменить выход из системы.

Наиболее быстрый способ добавить команду/пункт в меню *Пуск* - перетащить мышью значок программы на кнопку этого меню. Новый пункт меню расположится в верхней строке меню. Например, можно создать значок быстрого вызова для программы Norton Commander, разместив сначала на рабочем столе значок программы Norton Commander, а затем перетащив его мышью на кнопку *Пуск*.

Нередко приходится искать нужный файл/папку, так как забыто его имя или место расположения. Если известна папка, где расположен файл, то можно легко найти его по расширению. Сложнее, если не известно название и расширение. Чтобы быстро найти файл или папку на компьютере пользователя или на других компьютерах сети можно использовать команду *Файлы или Папки* из меню *Поиск*. Поиск можно выполнять по следующим критериям:

- по имени файла или папки и по цепочке символов, входящих в имя файла или папки;
- по расширению имени файла;
- по дате последней модификации;
- по размеру;
- по отрывку текста из документа или по заголовку какого-либо раздела.

Использование различных вкладок облегчает поиск файлов по определенным критериям.

Для быстрого вызова часто повторяемых команд можно воспользоваться динамическим меню, которое вызывается правой кнопкой мыши. Динамическое меню (рис. 2.8) содержит часто употребляемые команды. Набор команд зависит от выбранного объекта: значка диска, папки или файла, выделенного текста, панели задач или свободного места экрана.

Если щелкнуть правой кнопкой значок диска в окне программы *Мой компьютер*, то появятся команды: *Открыть, Проводник, Найти, Разделение, Форматировать, Вставить, Создать ярлык. Свойства*. Так же как и для диска, команды динамического меню для файла зависят от типа файла и дублируют меню *Файл*. Правую кнопку мыши удобно использовать для перемещения или копирования файла из одного окна в другое папки *Мой компьютер* или *Проводник*. После транспонирования значка папки/файла с нажатой правой кнопкой мыши появляется динамическое меню, позволяющее указать цель транспортировки: переместить или скопировать объект, создать значок быстрого вызова.

Папка *Мусорная корзина* предназначена для удаления ненужных файлов. Основное различие между выполнением команды *Удалить* в Windows'95 и в других программах состоит в том, что в новой версии операционной системы выбор команды приводит не к удалению файла, а к его перемещению в папку *Мусорная корзина*. Попавший в нее файл сохраняется до тех пор пока Корзина не будет «очищена». Чтобы удалить любой файл, папку или значок быстрого вызова, можно использовать команду *Удалить* или переместить значок удаляемого объекта мышью на значок *Мусорная корзина*. Значок перетаскиваемого объекта исчезнет. Чтобы просмотреть все файлы, находящиеся в Мусорной корзине, необходимо дважды щелкнуть ее значок. Появится окно папки со строкой меню, имеющим стандартный набор команд. Для восстановления файла, папки или значка быстрого вызова необходимо щелкнуть имя восстанавливаемого файла. Если надо восстановить несколько файлов, то имена файлов выделяют при нажатой клавише Ctrl. Затем используют команду *Восстановить* из меню *Файл*.

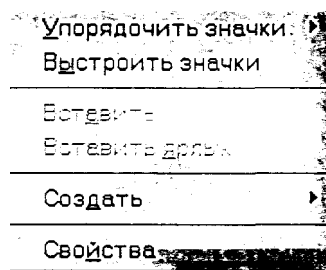


Рис. 2.8. Динамическое меню

В настоящее время для IBM PC-совместимых компьютеров выпускаются тысячи наименований винчестеров, адаптеров, контроллеров и других изделий. В ряде случаев их установка на компьютере вызывает значительные трудности с точки зрения совместимости, требует больших затрат времени пользователя для выбора положения перемычек. Компьютеры с Plug and Play - адаптерами не нуждаются в ручной установке DIP-переключателей. В соответствии с технологией Plug and Play после включения компьютера автоматически определяются названия вновь подключенных устройств или плат и их характеристики, выполняется их конфигурирование и автоматически загружаются соответствующие драйверы. Это может происходить и во время сеанса работы в случае изменения аппаратных средств компьютера. Аналогично фиксируется удаление какого-либо устройства или платы, выгружаются драйверы этого устройства с тем, чтобы они не занимали оперативную память и освободили системные ресурсы.

При инсталляции Windows'95 приложение Setup (Установка) определяет адаптеры и драйверы, которые не поддерживают новую технологию, и автоматически делает соответствующие записи в системных файлах. Даже если ваш компьютер не полностью поддерживает стандарт Plug and Play, Windows'95 поможет настроить аппаратное обеспечение с помощью диалогового окна *Свойства* с вкладкой *Ресурсы* рассматриваемого устройства. Для вызова окна *Устройства* сначала следует активизировать значок *Система* Панели Управления и двойным щелчком мыши выбрать устройство. В нижнем поле *Список конфликтующих устройств* указываются устройства, с которыми может конфликтовать рассматриваемое устройство. Рекомендуется установить флажок *Использовать автоматическую настройку*, чтобы операционная система сама производила настройку системных ресурсов.

Технология Plug and Play содержит три основных компонента: операционную систему, поддерживающую Plug and Play, Plug and Play BIOS и Plug and Play - устройства с соответствующими драйверами. Поэтому полное решение проблемы Plug and Play требует поддержки как на программном, так и на аппаратном уровнях.

В ходе работы с Windows'95 иногда приходится производить перезагрузку системы. Различают «холодную» и «горячую» перезагрузку компьютера. «Холодная» перезагрузка выполняется после выключения питания и его повторного включения. Необходимые для работы программы и параметры считываются в оперативную память заново. «Горячая» перезагрузка производится без выключения питания, одновременным нажатием клавиш Alt+Ctrl+Del. В этом случае вся информация, хранящаяся в оперативной памяти и не записанная на жесткий диск, стирается. «Горячую» перезагрузку, как правило, используют в том случае, когда программа «зависла» и не реагирует на нажатие на клавиши и на кнопки мыши. В ряде случаев при нажатии на клавиши Alt+Ctrl+Del появляется диалоговое окно *Снять Задачу*. Если окно позволит закрыть «зависшую» программу, то необходимость в перезагрузке компьютера отпадет.

Перед тем как выключать питание компьютера, необходимо закрыть все открытые документы и приложения. Выключение питания без закрытия документа может привести к потере данных, повреждению открытых файлов и трудностям с их открытием при последующих сеансах работы. После выключения компьютера без правильного выхода из системы возможны нарушения в логической структуре диска. Их исправить можно с помощью программы ScanDisk, расположенной в группе *Служебные программы*.

Для корректного выхода из Windows надо щелкнуть кнопку *Пуск* и команду *Завершить работу* в появившемся меню. Появится диалоговое окно *Завершение Работы*. Окно содержит кнопки-переключатели: *Выключить компьютер*, *Перезагрузить компьютер*, *Перезагрузить компьютер в режиме эмуляции MS DOS*, *Войти в систему под другим именем*. Все кнопки закрывают все программы. В нижней части окна Shut Down Windows расположены три кнопки - *Да*, *Нет*, *Справ-*

ка. Через небольшой промежуток времени после щелчка мышью кнопки *Да* компьютер будет подготовлен к выключению: будут очищены внутренние буферы и кэши дисков, обеспечено сохранение данных. Не следует выключать электропитание до тех пор, пока не появится сообщение: «Теперь питание компьютера можно выключить».

Мы ограничимся лишь приведенным выше кратким описанием принципов работы Windows. Реальное ее освоение (еще в большей мере, чем DOS и Norton Commander) - дело практических занятий за компьютером с помощью как многочисленных специальных руководств, так и встроенного справочника.

Контрольные вопросы и задания

1. Охарактеризуйте место операционных систем среди других видов программного обеспечения.
2. Каковы функции операционной системы?
3. Охарактеризуйте основные ступеньки эволюции операционных систем.
4. Каково содержание понятий
 - процесс?
 - ресурс?
 - виртуализация?
 - прерывание?
5. Охарактеризуйте функции основных компонент операционных систем.
6. В чем состоит назначение файловой системы ОС?
7. Что такое файл? Какие структуры файлов поддерживаются различными ОС?
8. Что такое каталог (директорий)? Для чего каталоги служат?
9. Какие операции над файлами обеспечиваются операционными системами?
10. Охарактеризуйте команды операционной системы MS DOS.
11. Опишите интерфейс оболочки ОС Norton Commander.
12. Охарактеризуйте стиль работы с помощью оболочки Windows.
13. По каким показателям Windows'95 превосходит Windows 3.11?
14. Опишите типичное окно Windows'95.
15. Какие функции выполняет меню Пуск? Поиск?
16. Как вызвать динамическое меню и каковы его возможности?

§ 2. ПОНЯТИЕ О СИСТЕМЕ ПРОГРАММИРОВАНИЯ

2.1. ОСНОВНЫЕ ФУНКЦИИ И КОМПОНЕНТЫ

Системы программирования - это комплекс инструментальных программных средств, предназначенный для работы с программами на одном из языков программирования. Системы программирования предоставляют сервисные возможности программистам для разработки их собственных компьютерных программ.

В настоящее время разработка любого системного и прикладного программного обеспечения осуществляется с помощью систем программирования, в состав которых входят

- трансляторы с языков высокого уровня;
- средства редактирования, компоновки и загрузки программ;
- макроассемблеры (машинно-ориентированные языки);
- отладчики машинных программ.

Системы программирования, как правило, включают в себя

- текстовый редактор (**Edit**), осуществляющий функции записи и редактирования исходного текста программы;
- загрузчик программ (**Load**), позволяющий выбрать из директория нужный текстовый файл программы;
- запускатель программ (**Run**), осуществляющий процесс выполнения программы;
- компилятор (**Compile**), предназначенный для компиляции или интерпретации исходного текста программы в машинный код с диагностикой синтаксических и семантических (логических)

ошибок;

- отладчик (**Debug**), выполняющий сервисные функции по отладке и тестированию программы;
- диспетчер файлов (**File**), предоставляющий возможность выполнять операции с файлами: сохранение, поиск, уничтожение и т.п.

Ядро системы программирования составляет язык. Существующие языки программирования можно разделить на две группы: процедурные и непроцедурные, рис. 2.9.

Процедурные (или алгоритмические) программы представляют из себя систему предписаний для решения конкретной задачи. Роль компьютера сводится к механическому выполнению этих предписаний.

Процедурные языки разделяют на языки низкого и высокого уровня.

Языки низкого уровня (машинно-ориентированные) позволяют создавать программы из машинных кодов, обычно в шестнадцатичной форме. С ними трудно работать, но созданные с их помощью высококвалифицированным программистом программы занимают меньше места в памяти и работают быстрее. С помощью этих языков удобнее разрабатывать системные программы, драйверы (программы для управления устройствами компьютера), некоторые другие виды программ.

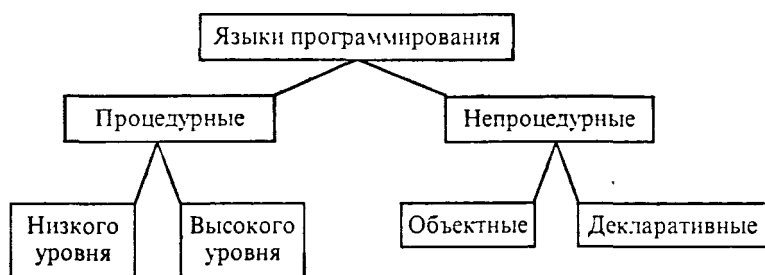


Рис. 2.9. Общая классификация языков программирования

Программы на языках высокого уровня близки к естественному (английскому) языку и представляют набор заданных команд.

Перечислим наиболее известные системы программирования.

1. Фортран (FORmula TRANslating system - система трансляции формул); старейший и по сей день активно используемый в решении задач математической ориентации язык.

2. Бейсик (Beginner's All-purpose Symbolic Instruction Code - универсальный символический код инструкций для начинающих); несмотря на многие недостатки и изобилие плохо совместимых версий - самый популярный по числу пользователей.

3. Алгол (ALGOrithmic Language - алгоритмический язык); сыграл большую роль в теории, но для практического программирования сейчас почти не используется.

4. ПЛ/1 (PL/I Programming Language - язык программирования первый). Многоцелевой язык; сейчас почти не используется.

5. Си (C - «си»); широко используется при создании системного программного обеспечения.

6. Паскаль (Pascal - назван в честь ученого Блеза Паскаля); чрезвычайно популярен как при изучении программирования, так и среди профессионалов. На его базе созданы несколько более мощных языков (Модула, Ада, Дельфи).

7. Кобол (COmmon Business Oriented Language - язык, ориентированный на общий бизнес); в значительной мере вышел из употребления.

8. Дельфи (Delphi) - язык объектно-ориентированного «визуального» программирования; в данный момент чрезвычайно популярен.

9. Джава (Java) - платформенно-независимый язык объектно-ориентированного программирования, чрезвычайно эффективен для создания интерактивных веб-страниц.

Среди непроцедурных языков наиболее известны

1. Лисп (Lisp);
2. Пролог (PROgramming in LOGic);
3. Оккам (назван в честь философа У. Оккама).

Широкое распространение среди разработчиков программ, а также при обучении програм-

мированию, получили системы программирования «Турбо» (Turbo) фирмы Borland, ядром которых являются трансляторы с языков программирования Бейсик, Паскаль, Си, Пролог и др. Интерфейс Турбо-оболочки для любых систем программирования внешне совершенно одинаков и предоставляет пользователю стандартный набор функций и команд, описанных выше и отображаемых в главном меню системы.

Рассмотрим технологию разработки программ с использованием популярной системы программирования Турбо-Паскаль 7 (оставив знакомство с самим языком до следующей главы).

В подобных интегрированных системах программирования сделана попытка предоставить разработчику программ максимум сервисных возможностей. Помимо основных функций система Турбо-Паскаль 7 позволяет настроить компилятор на работу в трех режимах: обычном режиме MS DOS (Real), защищенном режиме (Protected) и в режиме операционной среды Windows (Windows).

После загрузки системы (файл TURBO. EXE), на экране монитора появляется интерфейсное окно, рис. 2.10.

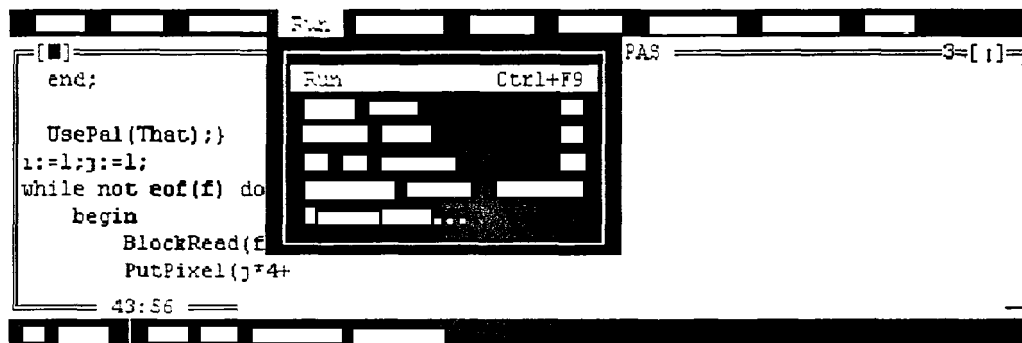


Рис. 2.10. Вид экрана интегрированной среды Турбо-Паскаля версии 7 (монтаж)

Главное меню системы (верхняя строка экрана) содержит команды, которые позволяют осуществлять следующие виды работ:

- File - работа с файлами (сохранение, загрузка, связь с операционной системой);
- Edit - работа с текстовым редактором (после загрузки системы по умолчанию текстовый редактор находится в активном состоянии);
- Search - поиск и замена фрагментов текста;
- Run - запуск программы на выполнение;
- Compile - компиляция программы и установка параметров компиляции;
- Debug - установка параметров отладки программы;
- Tools - инструментальные программные средства (ненавязчивый сервис);
- Options - установка опций интегрированной среды;
- Window - работа с окнами;
- Help - система помощи и подсказок.

Для начала работы с системой программирования необходимо иметь проект текста программы, который можно набирать на рабочем поле окна системы. Встроенный текстовый редактор прост и максимально приспособлен для набора текстов программ на языке Паскаль. В нем предусмотрена специальная подсветка управляющих структур, команд. Удобна система контекстной помощи (Shift+F1), которая вызовет подсказку по набираемому текущему тексту программы в любой момент и в любом месте. Впрочем, текст программы можно приготовить в любом текстовом редакторе, хранящем тексты в ASCII-кодах (например, в Лексиконе); необходимо лишь снабдить имя файла расширением .pas.

Если текст (тексты) программы был ранее сохранен на жестком диске или дискете, то он может быть загружен в поле редактирования с помощью пункта меню **File**.

После окончания формирования текста необходимо откомпилировать программу (пункт меню **Compile**). Если в программе есть ошибки, то компилятор их укажет. После исправления ошибок можно снова повторить компиляцию.

После удачной компиляции запуск программы осуществляется командой меню **Run**.

Но на этом этапе чаще всего работа не заканчивается. Сложные алгоритмы требуют тестирования и отладки. Многие программы состояются из отдельных модулей, требуют связи с дру-

гими программами и системами и т.д. Для решения всех этих проблем предназначены другие команды системы (**Debug**, **Options** и пр.).

Разумеется, программисту, работающему на Паскале, нет нужды самому программировать такие непростые, но часто встречающиеся операции, как вычисление значений математических функций, построение изображений простых геометрических объектов (отрезков прямых, окружностей и т.д.), очистка экрана и множество других. Высокоэффективные, тщательно отлаженные программы таких действий сведены в стандартные модули и надо лишь уметь к ним обратиться. В состав пакета библиотек стандартных модулей входят: **Crt** - работы с экраном, **Graph** - работы с графикой и другие, такие как **Overlay**, **String**, **System**, **Turbo3**, **WinAPI**, **WinCrt**, **WinDos**, **WinPrn**, **WinTypes**, **WinProcs**.

2.2. ТРАНСЛЯЦИЯ ПРОГРАММ И СОПУТСТВУЮЩИЕ ПРОЦЕССЫ

С появления первых компьютеров программисты серьезно задумывались над проблемой кодирования компьютерных программ. Уже с конца 40-х годов стали появляться первые примитивные языки программирования высокого уровня. В них программист записывал решаемую задачу в виде математических формул, а затем, используя специальную таблицу, переводил символ за символом, преобразовывал эти формулы в двухлитерные коды. В дальнейшем специальная программа (впоследствии названная интерпретатором) превращала эти коды в двоичный машинный код. Первый компилятор был разработан Г. Хоппер в начале 50-х годов; он осуществлял функцию объединения команд и в ходе трансляции производил организацию подпрограмм, выделение памяти компьютера, преобразование команд высокого уровня (в то время псевдокодов) в машинные команды. В дальнейшем компиляторы и интерпретаторы для языков Ассемблера стали развиваться и прочно вошли в практику компьютерного дела.

Идеи трансляции (перекодирования) одних символов в другие легли в основу создания различных языков программирования с соответствующими трансляторами - компиляторами и/или интерпретаторами. Отличие компиляторов от интерпретаторов заключается в процедуре трансляции текста в машинный код. Компилятор преобразует весь текст программы в последовательный набор машинных команд, который в дальнейшем отправляется на выполнение (пример компилятора с языка Паскаль). Интерпретатор же осуществляет трансляцию по принципу синхронного перевода. Каждая отдельная строка программного текста транслируется, а затем, после ее интерпретации, команды этой строки выполняются (пример языка Бейсик). Современные трансляторы с языков программирования высокого уровня, систем управления базами данных интегрируют в себе возможности и достоинства компиляторов и интерпретаторов, а в системы программирования добавляют различные сервисные утилиты по трансляции и отладке создаваемых программ.

Важнейшим элементом в развитии систем программирования выступили подпрограммы. Появление аппарата подпрограмм существенно облегчило процесс разработки системных и прикладных программ. Подпрограммы позволили формировать библиотеки из наиболее часто употребляемых в программах алгоритмов -процедур и функций. В системах программирования обязательно присутствуют стандартные (встроенные в систему) библиотеки подпрограмм. Например, в их число входят подпрограммы вычисления математических функций $\sin(x)$, $\cos(x)$, $\text{abs}(x)$ и др.

В настоящее время распространены пользовательские и прикладные библиотеки подпрограмм. Их число увеличивается. Меняется структура библиотечных подпрограмм. В современных языках получили распространение модули (**Unit**), представляющие специализированные пакеты взаимосвязанных подпрограмм определенного назначения, например по работе с клавиатурой, с графикой и пр. Развитие объектно-ориентированного программирования позволило создавать библиотеки объектов и подпрограмм с объектными типами данных (**Object**). Примером могут служить оболочки типа **TurboVision**.

Современная программа представляет набор команд, операторов и выражений, в которых имеются ссылки (прямые или косвенные) на различные подпрограммы из существующих в системе программирования библиотек, модулей, объектов. В этой связи исходный текст программы, как правило, занимает по объему места в памяти в несколько раз меньше, чем его оттранслированный вариант в машинных кодах. Как это происходит?

Рассмотрим один из вариантов трансляции программы с языка программирования Паскаль. Исходный текст программы решения квадратного уравнения представлен ниже:

```

program KvadUrn;
  var A, B, C, D, X1, X2: REAL;
begin
  writeln;
  writeln( 'введи A,B,C' ); read(A,B,C);
  D:=B*B-4*A*C;
  if D<0 then write('корней нет')
    else begin
      X1:=(-B+sqrt(D))/(2*A);
      X2:=(-B-sqrt(D))/(2*A);
      write('X1=', X1, ' X2=', X2);
    end
end.

```

Предположив, что этот текст (по отношению к процессу трансляции выступающий как **исходный** модуль) сформирован одним из текстовых редакторов, попытаемся отправить его на выполнение. Прежде всего его необходимо перевести в машинный двоичный код (называемый **абсолютным** или **загрузочным модулем**). Для этого на первых этапах осуществляется трансляция (в данном случае, как это реализовано в системах программирования Паскаля, компиляция) исходного текста в машинный код (**объектный модуль**). Однако, объектный модуль не может быть использован для выполнения программы, поскольку в нем нет программ по выполнению процедур ввода (read) и вывода (write, writeln), а также вычисления функции извлечения квадратного корня (sqrt). В исходном тексте программы ссылки на указанные библиотечные подпрограммы отмечены знаком `{*}`.

Следующий шаг трансляции - компоновка - заключается в подключении к исходному объектному модулю объектных модулей соответствующих подпрограмм в места ссылок на них (исходные тексты этих подпрограмм в системе вообще отсутствуют). Другими словами, на место процедуры Write помещается подпрограмма, осуществляющая процедуру вывода данных на экран дисплея. Таким образом после компоновки (или, иначе, редактирования связей link editor) возникает абсолютный модуль, намного превышающий по объему размер исходного текста программы. Он и является исполняемым компьютером после его запуска. Расширениями его файлового имени, как правило, являются .com или .exe.

В силу того, что объектные модули не предназначены для непосредственного исполнения, в них обычно нет привязки составляющих их машинных команд к конкретному месту в ОЗУ. Адреса машинных слов бывают условными, что помогает компоновщику размещать объектные модули в свободных местах ОЗУ (заменяя условные адреса команд на конкретные).

Многие системы программирования дополнительно содержат промежуточные этапы трансляции. В этих системах на первом шаге предусмотрена трансляция исходного текста в макроассемблерный код, а затем в объектный модуль. Это связано с историей развития языков программирования, а также с тем, что многие подпрограммы удобнее писать на языке Ассемблера, и подключать их легче на этапе линкования ассемблерного модуля с ассемблерными библиотеками подпрограмм.

В современных системах программирования, например, Турбо-Паскаль, Турбо-Си весь этот сложный процесс трансляции с компоновкой подпрограмм скрыт от пользователя и осуществляется специальными компиляторами.

Коротко об отладчиках. Эти программы входят в современные системы программирования и предоставляют средства для просмотра и изменения значений переменных в ходе отладки программы, поиска ошибок и т.д. Использование отладчиков значительно облегчает процесс доводки больших программ.

Заметим, что описанный процесс трансляции характерен для компиляции. Последовательно реализованный интерпретатор объектного модуля фактически не создает. В этом его и недостаток, и достоинство (экономия машинной памяти). Впрочем, у современных ЭВМ, в том числе и персональных, проблема малого ОЗУ отходит на второй план, и интерпретация встречается все реже, так как эффективность этого процесса в целом значительно ниже.

Остается непонятным, как детально происходит трансляция. Пользователь может не уметь сам вручную оттранслировать программу (даже столь короткую, как вышеприведенная), но эле-

ментарное понимание этого сложного процесса необходимо.

На первом этапе транслятор производит синтаксический анализ исходной программы - проверяет, не нарушены ли формальные правила, содержащиеся в данном языке программирования. Например, в Паскале текст может встретиться либо внутри текстовой константы (т.е. в апострофах), либо внутри комментария. Если такой текст встретился в другом месте, то это явная ошибка. В системе программирования встроены описания всех синтаксически разрешенных конструкций, и транслятор их применяет к исходной программе. Для задания синтаксиса применяются формы Бэкуса-Наура и синтаксические диаграммы, о которых будет рассказано в следующей главе.

Первой фазой синтаксического анализа является лексический анализ. Он заключается в просмотре литер исходной программы и построении из них лексически допустимых единиц - идентификаторов, ключевых слов языка, чисел и т.д. Во второй фазе эти единицы уже рассматриваются как неделимые и проверяется допустимость их сочетания.

Даже если в синтаксическом смысле исходная программа верна, это не означает, что она имеет смысл в рамках данного языка программирования. На следующем этапе семантического анализа транслятор ищет ошибки такого рода: числа употребления слов BEGIN и END не совпадают; переменные не описаны (в языке, требующем обязательного явного описания переменных), т.е. текст программы *непонятен* (семантика - смысловая сторона языка).

Лишь после того, как в программе все синтаксически правильно и семантически понятно, транслятор переводит операторы программы в машинный код. Это отнюдь не означает, что в программе все благополучно - не исключены ошибки этапа исполнения (деление на ноль, выход за границу массива, переполнение разрядов и т.д.).

Различные фазы компиляции могут быть как последовательными, так и частично перекрывающимися во времени. В зависимости от способа реализации компилятор читает и обрабатывает исходный текст один или несколько раз, называясь соответственно однопроходным, двухпроходным и т.д.

Контрольные вопросы

1. Назовите минимальный состав системы программирования, необходимый для разработки программы.
2. Какие имеются сравнительные преимущества и недостатки у компиляторов и интерпретаторов?
3. Назовите основные этапы трансляции программы.

§3. ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ОБЩЕГО НАЗНАЧЕНИЯ

3.1. КЛАССИФИКАЦИЯ

Прикладные программы предназначены для того, чтобы обеспечить применение вычислительной техники в различных сферах деятельности человека. Помимо создания новых программных продуктов разработчики прикладных программ большие усилия тратят на совершенствование и модернизацию популярных систем, создание их новых версий. Новые версии, как правило, поддерживают старые, сохраняя преемственность, и включают в себя базовый минимум (стандарт) возможностей.

Один из возможных вариантов классификации программных средств (ПС), составляющих прикладное программное обеспечение (ППО), отражен на рис.2.11. Как и почти всякая классификация, приведенная на рисунке не является единственно возможной. В ней представлены даже не все виды прикладных программ. Тем не менее, использование классификации полезно для создания общего представления о ППО.

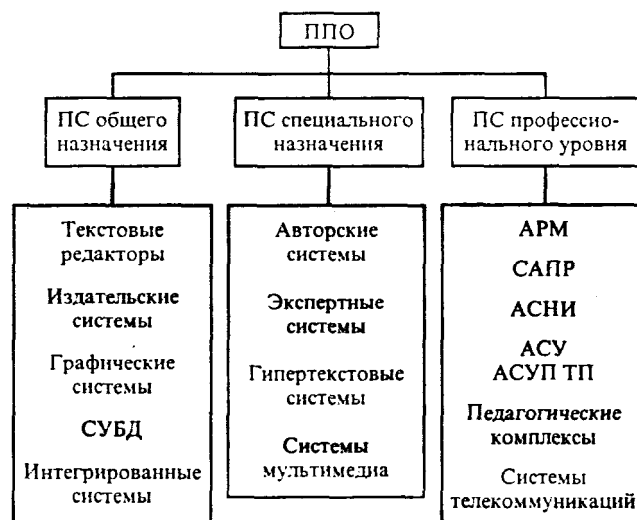


Рис.2. II. Классификация прикладного программного обеспечения

3.2. ИНСТРУМЕНТАЛЬНЫЕ ПРОГРАММНЫЕ СРЕДСТВА ОБЩЕГО НАЗНАЧЕНИЯ

Несмотря на широкие возможности использования компьютеров для обработки самой разной информации, самыми популярными являются программы, предназначенные для работы с текстами - текстовые редакторы и издательские системы. Текстовыми редакторами называют программы для ввода, обработки, хранения и печатания текстовой информации в удобном для пользователя виде. Эксперты оценивают использование компьютера в качестве печатающей машинки в 80%.

Большую популярность приобрели программы обработки графической информации. Компьютерная графика в настоящее время является одной из самых динамично развивающихся областей программного обеспечения. Она включает в себя ввод, обработку и вывод графической информации - чертежей, рисунков, картин, текстов и т.д. - средствами компьютерной техники. Различные типы графических систем позволяют быстро строить изображения, вводить иллюстрации с помощью сканера или видеокамеры, создавать анимационные ролики.

Графические редакторы позволяют пользоваться различным инструментарием художника, стандартными библиотеками изображений, наборами стандартных шрифтов, редактированием изображений, копированием и перемещением фрагментов по страницам экрана и др. Для выполнения расчетов и дальнейшей обработки числовой информации существуют специальные программы - электронные таблицы. В процессе деятельности любого специалиста часто требуется представить результаты работы в виде таблиц, где одна часть полей занята исходными данными, а другая -. результатами вычислений и графического анализа. Характерными для них является большой объем перерабатываемой информации, необходимость многократных расчетов при изменении исходных данных. Автоматизацией подобной рутинной работы и занимаются электронные таблицы.

Одним из наиболее перспективных направлений развития вычислительной техники является создание специальных аппаратных средств для хранения гигантских массивов информационных данных, и последующей нечисловой обработки их -поиска и сортировки. Для компьютерной обработки подобных **баз данных** используют **системы управления базами данных**. СУБД - это набор средств программного обеспечения, необходимых для создания, обработки и вывода записей баз данных. Различают несколько типов СУБД: *иерархические, сетевые, реляционные*. При работе с СУБД выделяют несколько последовательных этапов:

- проектирование базы данных;
- создание структуры базы данных;
- заполнение базы данных;
- просмотр и редактирование базы данных;
- сортировку базы данных;
- поиск необходимой записи;
- выборку информации;

- создание отчетов.

Как правило, большинство популярных систем управления базами данных поддерживают эти этапы и предоставляют удобный инструментарий для их реализации.

Желание объединить функции различных прикладных программ в единую систему привело к созданию интегрированных систем. Универсальные *интегрированные системы* разрабатывались по принципу единой системы, содержащей в качестве элементов текстовые и графические редакторы, электронные таблицы и систему управления базами данных. Примеры: Framework, Works, Мастер. Современная концепция интеграции программных средств - кооперация отдельных прикладных программных систем по типу широко известного пакета MicroSoft Office. Сами системы, входящие в пакет, являются независимыми, более того, они сами представляют локально интегрированный пакет, поскольку помимо основной своей задачи поддерживают функции других систем. Например, текстовый редактор Word обладает возможностью манипулировать с электронными таблицами и базами данных, а в электронной таблице Excel встроен мощный текстовый редактор. Для сопряжения информационных данных из различных программных систем в них предусматривают импорт-экспортную систему обмена с перекодировкой форматов представления данных.

3.3. ИНСТРУМЕНТАЛЬНЫЕ ПРОГРАММНЫЕ СРЕДСТВА СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ

Разработчики создают специальные программные системы целевого назначения для специалистов в некоторой предметной области. Такие программы называют авторскими инструментальными системами. **Авторская система** представляет интегрированную среду с заданной интерфейсной оболочкой, которую пользователь может наполнить информационным содержанием своей предметной области.

Экспертная система - это программа, которая ведет себя подобно эксперту в некоторой узкой прикладной области. Экспертные системы призваны решать задачи с неопределенностью и неполными исходными данными, требующие для своего решения экспертных знаний.

Кроме того, эти системы должны уметь объяснять свое поведение и свое решение.

Принципиальным отличием экспертных систем от других программ является их адаптивность, т.е. изменчивость в процессе самообучения.

Принято выделять в экспертных системах три основных модуля:

- модуль базы знаний;
- модуль логического вывода;
- интерфейс с пользователем.

Экспертные системы, являющиеся основой искусственного интеллекта, получили широкое распространение в науке (классификация животных и растений по видам, химический анализ), в медицине (постановка диагноза, анализ электрокардиограмм, определение методов лечения), в технике (поиск неисправностей в технических устройствах, слежение за полетом космических кораблей и спутников), в политологии и социологии, криминалистике, лингвистике и т.д.

В последнее время широкую популярность получили программы обработки гипертекстовой информации. Гипертекст – это форма организации текстового материала не в линейной последовательности, а в форме указании возможных переходов (ссылок), связей между отдельными его фрагментами. В обычном тексте используется обычный линейный принцип размещения информации и доступ к нему осуществляется последовательно. В **гипертекстовых системах** информация напоминает текст энциклопедии, и доступ к любому выделенному фрагменту текста осуществляется произвольно по ссылке. Организация информации в гипертекстовой форме используется при создании справочных пособий, словарей, контекстной помощи (Help) в прикладных программах.

Расширение концепции гипертекста на графическую и звуковую информацию приводит к понятию гипермедиа. Идеи **гипермедиа** получили распространение в сетевых технологиях, в частности в Интернет-технологиях. Технология WWW (World Wide Web) позволила структурировать громадные мировые информационные ресурсы посредством гипертекстовых ссылок. Появились программные средства, позволяющие создавать подобные Web-странички. Стали развиваться механизмы поиска нужной информации в лабиринте информационных потоков. Популярными поисковыми средствами в Интернет являются Yahoo, AltaVista, Magellan, Rambler и др.

Мультимедиа (multimedia) - это взаимодействие визуальных и аудиоэффектов под управ-

лением интерактивного программного обеспечения. Появление и широкое распространение компакт-дисков (CD-ROM) сделало эффективным использование мультимедиа в рекламной и информационной службе, сетевых телекоммуникационных технологиях, обучении.

Мультимедийные игровые и обучающие системы начинают вытеснять традиционные «бумажные библиотеки». Сегодня в библиотеках CD-ROM можно «гулять» по музеям, Московскому Кремлю и т.д. с помощью «электронного путеводителя».

3.4. ПРОГРАММНЫЕ СРЕДСТВА ПРОФЕССИОНАЛЬНОГО УРОВНЯ

Каждая прикладная программа этой группы ориентируется на достаточно узкую предметную область, но проникает в нее максимально глубоко. Так функционируют **АСНИ** - автоматизированные системы научных исследований, каждая из которых «привязана» к определенной области науки, **САПР** - системы автоматизированного проектирования, каждая из которых также работает в узкой области, **АСУ** - автоматизированные системы управления (которых в 60 - 70 годах были разработаны тысячи).

Наконец, еще раз подчеркнем не только условность предложенной выше классификации, но и наличие пересечений. Так, каждую конкретную экспертную систему вполне можно отнести к ППО профессионального уровня; принцип гипертекста реализован в ряде авторских систем и т.д.

3.5. ОРГАНИЗАЦИЯ «МЕНЮ» В ПРОГРАММНЫХ СИСТЕМАХ

Прикладные программы нацелены на широкий круг пользователей (непрограммистов) и предполагают диалоговый режим работы человека с компьютером. Широкой популярностью пользуются программы, обладающие дружественным интерфейсом, т.е. таким, который не требует от пользователя больших усилий в работе со всеми необходимыми периферийными устройствами, специальных настроек компьютера и обладает удобной системой управления и диалога.

Интерактивный режим в прикладных программах осуществляется по двум принципам: «смотри и выбирай» и «подтверждай то, что я делаю». Программы не утрачивают работоспособности при ошибках пользователя, позволяют легко и безболезненно исправлять ошибочные действия путем их отмены, а также обращаться в любой момент к контекстной помощи. Все принципы дружественного интерфейса реализуются специальной системой интерактивного (диалогового) общения компьютерной программы и пользователя, называемой «пользовательским меню» или просто «меню».

«Меню» представляет набор команд, указаний и данных, который в любой момент доступен пользователю для выбора дальнейшего действия. Указатель (курсор) имеет возможность циклически сканировать меню и управляется клавишами со стрелками и/или манипулятором «мышь». Выбор команды осуществляется установкой курсора на его пункт и нажатием клавиши ввода <Enter>, или указанием стрелки «мыши» и двойным щелчком ее клавиши. Более быстрый выбор команды может быть осуществлен нажатием специальных клавиш или их комбинацией. Как правило, в них участвуют клавиши с буквой, с которой начинается название команды. По своей организации меню представляет иерархическую структуру с системой вложенных подменю («выплывающие», «ниспадающие», «оконные» и пр.) с возможностью возврата из любого пункта в главное (основное) меню.

Меню бывает текстовым и/или графическим с комментариями по каждому своему пункту. Прикладные программы дополнительно имеют функциональное клавишное меню для быстрого выполнения каких-либо команд («горячие клавиши» - «hot key»). Например, функциональная клавиша F1 чаще используется для экстренного вызова справочной информации (Help - помощь), клавиша F2 - для сохранения данных во внешней памяти.

Фирмы-разработчики программных средств организуют программные меню по своим стандартам и единообразно. Так, например, фирма «Борланд» практикует свой фирменный интерфейс, который легко распознается всеми программистами и пользователями, работающими в системах программирования Турбо: Турбо-Паскаль, Турбо-Бейсик, Турбо-Си и т.п. (см. выше рис.2.10). В верхней части экрана дисплея в Турбо-программах помещается горизонтально главное меню, каждый пункт которого может иметь ниспадающее подменю. В нижней части помещаются команды для функциональных клавиш («горячие» клавиши), рядом - строка статуса, которая дает коммен-

тарии к выбираемым командам меню и некоторые параметры состояния прикладной программы.

Таким образом, меню - это некоторый перечень команд (функций), которые имеются в распоряжении пользователя на различных этапах работы с программной системой.

Исторически первым видом «меню» можно считать перечень команд в виде пронумерованного списка возможных функций

Например:

1 - редактирование текста;

2-трансляция программы;

3 - выход.

Оставалось лишь выбрать режим путем нажатия клавиши с его номером и клавиши <Ввод>.

Другой вариант такого меню - нумерованный список команд, в котором выбор нужной команды осуществляется нажатием клавиши с первой буквой ее имени.

Например:

Редактирование текста;

Трансляция программ;

Выход.

Еще один схожий способ, бывший ранее популярным - наличие справа или слева от списка стрелки, движение которой по вертикали управляется клавишами ↓ и ↑. Установив стрелку против нужной строки и нажав клавишу <Ввод>, осуществляем выбор нужной команды.

Например:

Редактирование текста;

Трансляция программ; <=

Выход.

Подобный вид меню определялся уровнем развития аппаратной и программной частей компьютера и вынужденной необходимостью ориентации на алфавитно-цифровые (символьные) дисплеи.

Дальнейшее развитие числовых меню привело к символьным меню, в которых выбор заданной функции осуществлялся с помощью ввода соответствующего символа или с помощью функциональных клавиш клавиатуры F1, F2, F3 ...

Принципиальным шагом в организации эффективных пользовательских меню стало использование графических средств. Появилась возможность создать указатель (в разных программных средах - светящийся курсор, стрелка, выделенный прямоугольник и т.п.), перемещающийся с помощью клавиш со стрелками, а выбор пункта меню осуществлять нажатием специальной клавиши, как правило, ENTER или <Пробел>.

Один из признаков дружелюбности меню - когда подведение указателя к некоторому элементу меню отображает комментарий функционального назначения. Например, в пункте *Текст* может появиться комментарий:

Операции над текстом в целом: считывание, запись, печать

Если выбрать этот пункт меню нажатием клавиши ввода ENTER, то появится подменю команд работы с текстом.

Подобный принцип иерархии в построении меню, который включает главное (основное меню) и дерево подменю, позаимствован из организации структуры директорий (каталогов) файловой системы компьютера. Теперь появилась возможность строить не только дружелюбный интерфейс, но и дизайн. Возникли меню с системой «ниспадающих», «всплывающих», «многооконных» и т.д. подменю.

Современные типы меню строятся с использованием графического и символьного режимов. Символьный принцип в меню используют для выбора быстрых команд. Соответствующим командам назначаются клавиши, их комбинации или функциональные клавиши F1 ... F 12.

Существует определенная традиция действий «горячих» клавиш. В частности, в большин-

стве программ клавиша F2 сохраняет результаты работы, комбинация ALT+X осуществляет выход из программы и т.д.

Значительные удобства пользователю предоставляет специальный манипулятор «мышь», который позволил серьезно облегчить и предоставить комфортные условия работы. Перемещая с помощью мыши указатель, можно одним нажатием кнопки мыши вызвать требуемую функцию.

Современные программные системы построены на интерактивных меню, использующих принцип «кнопки», которые требуют от пользователя минимальных знаний и действий. В них закладывается удобный и оптимальный для работы человека диалоговый режим. Меню содержат интерактивные формы:

- с шаблоном ответа

Продолжить? Y/N

- со справочником ответа

Какой цвет назначить: черный белый красный голубой

- с назначением параметров, рис. 2.12;
- многостраничные формы, рис. 2.13.

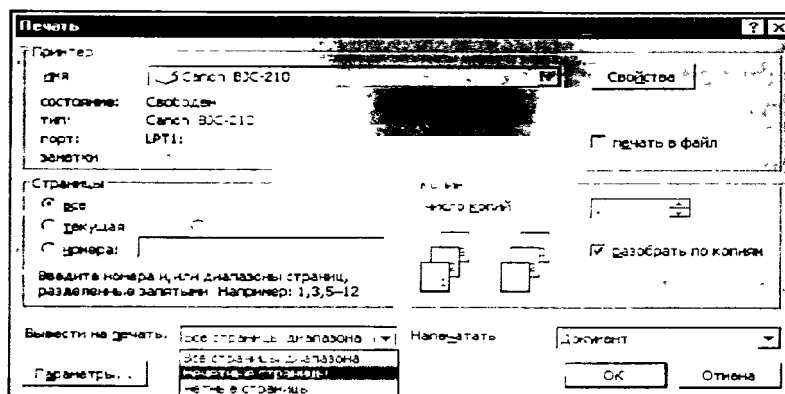


Рис. 2.12. Пример меню с назначением параметров (меню формы печати в Word)

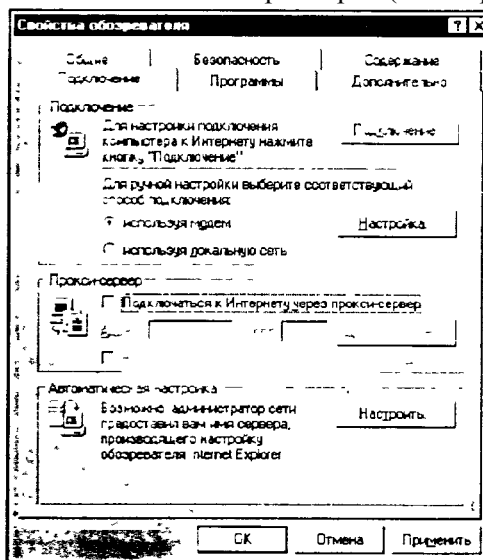


Рис.2.13. Пример многостраничного меню (свойства обозревателя Internet Explorer 4.0)

Интерфейс современных меню строится таким образом, чтобы запрос-ответ пользователя был однозначен, прост и удобен. В силу большой значимости систем организации меню, в большинстве инструментальных сред для разработки программ предусматриваются специальные процедуры и объекты создания меню. Особенное внимание им уделяется в системах управления базами данных (СУБД), авторских системах, в объектно-ориентированных языках программирования.

При работе с прикладными программами пользователю приходится сталкиваться в первую очередь с меню. От того, как он освоил работу с его пунктами, зависит эффективность использования информационной технологии.

1. Что такое текстовые редакторы? Для чего они нужны?
2. Что такое графические редакторы? Для чего они нужны?
3. Что такое электронные таблицы? Для чего они нужны?
4. Что такое СУБД? Для чего они нужны?
5. Что такое интегрированные системы? Для чего они нужны?
6. Что такое экспертные системы? Для чего они нужны?
7. Что такое авторские системы? Для чего они нужны?
8. Что такое гипертекст, гипермедиа?
9. Что такое мультимедиа?
10. Какие бывают типы меню?
11. Для чего в меню используют «горячие» клавиши?
12. Разработайте программу «меню» на одном из языков программирования.

§ 4. СИСТЕМЫ ОБРАБОТКИ ТЕКСТОВ

4.1. ЭЛЕМЕНТЫ ИЗДАТЕЛЬСКОГО ДЕЛА

Для того, чтобы уверенно работать с текстовыми редакторами и настольными издательскими системами, необходимо освоить и уяснить некоторые сведения из издательского дела.

Особую значимость при подготовке и формировании текста для издания имеют шрифты. **Шрифты** - основное изобразительное средство издательских систем, с их помощью можно добиться большой художественной выразительности текста.

Шрифты различают по *гарнитуре* (рисунку), *начертанию*, *размеру* и *назначению*. Гарнитурой называется совокупность шрифтов одного рисунка во всех начертаниях и кеглях. Кегль - размер шрифта, определяемый размером литеры по верикали, исчисляемый в *пунктах* (1 пункт равен 0,367 мм) Полный комплект гарнитуры содержит шрифты всех начертаний и кеглей, а в каждом кегле - русский и латинский (и, если нужно, другие) алфавиты прописных и строчных букв, а также относящиеся к ним знаки.

Различия между буквами разных шрифтов объясняется их различным построением. Среди элементов из которых строятся буквы выделяют:

- основные штрихи (задают структуру буквы);
- дополнительные штрихи (играют вспомогательную и соединительную роль);
- засечки;
- верхние и нижние выносные элементы;
- овалы и полуovalы (с наплывами или без них);
- концевые элементы.

Буквы располагаются по *базовой линии*. Расстояние между строками называют *интерлиньяжем*. Отношение толщины основных и дополнительных элементов определяет контрастность шрифта. Различают неконтрастные, малоконтрастные, контрастные и очень контрастные шрифты. Форму букв шрифта определяют *цветность* и *ритм* (соотношение черного и белого, просветы, наплывы и пр.). Отношение высоты буквы к ее ширине называют шириной шрифта. Бывают сверхузкие, узкие, нормальные, широкие и сверхширокие шрифты. *Насыщенность* шрифта определяется *светлотой*. Бывают сверхсветлые, светлые, нормальные, полужирные, жирные и сверхжирные шрифты. Шрифты могут быть прямыми и наклонными. Наклонный вариант шрифтов часто называют *курсивом*. Шрифты одного типа, но отличающиеся по ширине, насыщенности и наклону являются различными *начертаниями* одного и того шрифта. Начертания одного шрифта составляют *гарнитуру*.

Шрифт на компьютере - это файл или группа файлов, обеспечивающих вывод текста на печать со стилизованными особенностями шрифта. Существуют программы, позволяющие создавать собственные варианты шрифтов. Как правило, такие программы входят в состав текстовых редакторов и издательских систем. Однако, существует довольно большой спектр стандартных (в компьютерном смысле) шрифтов, разработанных полиграфистами. Приведем некоторые из них:

Akademь	Аа Бб Вв Гг Дд 1 2 3 4 5 6 7 8 9 0
Arial Cyr	Аа Бб Вв Гг Дд 1 2 3 4 5 6 7 8 9 0
Baltika	Аа Бб Вв Гг Дд 1 2 3 4 5 6 7 8 9 0
Courier New Cyr	Аа Бб Вв Гг Дд 1 2 3 4 5 6 7 8 9 0
DEJDEJONA	АА АА АА АА АА 1234567890

и др. Основной текст данной главы набран шрифтом Times New Roman Cyr. Наиболее часто используются следующие гарнитуры:

1. *Литературная* - для набора всех видов изданий, кроме букварей, энциклопедий, а также малоформатных, нормативных, периодических и литературно-художественных изданий;
2. *Обыкновенная новая* - для набора всех видов книжно-журнальных изданий, кроме учебников для школы, карманных и нормативных изданий;
3. *Банниковская* и *Академическая* - для набора литературно-художественных, научных, учебных и научно-популярных изданий по гуманитарным областям знаний;
4. *Школьная* - для набора учебников начальной и средней школы, изданий для детей, художественной и научно-популярной литературы, нормативных изданий и массовых иллюстрированных журналов;
5. *Балтика* и *Таймс* - для набора художественной и научно-популярной литературы, вузовских учебников.

Каждый текст, подготовленный к изданию в качестве брошюры или книги, должен пройти техническое редактирование, которое предполагает подготовку *оригинал-макета* готового издания. Издательским текстовым оригинал-макетом является набранный, сверстаный на компьютере и отпечатанный на лазерном (или струйном) принтере текстовый оригинал, представляющий собой точный прообраз будущего издания (по числу страниц, абзацев, рисунку шрифта), предназначенный для изготовления печатной формы. Исходным материалом для подготовки оригинал-макета является отредактированный, вычитанный автором и литературным редактором размеченный текстовый оригинал. Техническое редактирование - процесс достаточно сложный, особенно если в тексте есть таблицы, схемы, формулы и иллюстрации.

4.2. ТЕКСТОВЫЕ РЕДАКТОРЫ

Программы-текстовые редакторы предназначены для подготовки текстовых материалов на компьютере.

Поскольку текстовые материалы бывают различной сложности по набору и верстке, существуют и развиваются различные программы обработки текстов. Их можно классифицировать по уровням требований к обработке текстов. Перечислим наиболее популярные из них.

А. Программы для набора и обработки простых текстов:

Lexicon
Refis
Norton Editor
MultiEdit
Word 4.0 и 5.0 и др.

Б. Программы для набора сложных текстов:

ChiWriter
TechWord
Word 5.5, 6.0 и 7.0
Word for Windows 1.1 и 2.2
WordPerfect
TeX, LaTeX
и др.

Каждый текстовый редактор имеет свои технические требования к составу и конфигурации компьютерной техники.

Для набора простого текста достаточно иметь

- персональный компьютер IBM PC/XT, AT (или совместимые с ними) с емкостью оперативной памяти 640 кбайт;

- дисплей монохромный CGA, EGA или Hercules 14.

Для набора сложных текстов, содержащих формулы, таблицы, схемы необходимо иметь

- персональный компьютер IBM PC/AT 286 (или совместимые с ним), с емкостью оперативной памяти 1 Мбайт;

- дисплей монохромный CGA, EGA или Hercules 14;

- накопитель на жестком магнитном диске (30 - 40 Мбайт);

- матричный 9- или 24-игольчатый принтер, струйный или лазерный принтер;

- мышь.

Для подготовки черно-белых иллюстраций, логотипов, обложек, титулов, разработки шрифтов необходимо иметь

- персональный компьютер IBM PC/AT 386/486 (или совместимые с ним) с емкостью оперативной памяти 4 Мбайт;

- дисплей цветной VGA, SVGA 14/15;

- накопитель на жестком магнитном диске (более 80 Мбайт);

- лазерный принтер;

- сканер;

- мышь.

Разумеется, подходят не только компьютеры IBM-совместимые, но и другие (например, Macintosh); разные модели IBM указаны лишь поскольку они наиболее распространены для ориентации по уровню требований.

Разработчики программ текстовых редакторов стараются предусмотреть в них предоставление пользователю всех необходимых операций и сервисных возможностей для эффективной обработки текстов. Выделим главные из них:

- набор текста в интерактивном режиме;
- редактирование текста;
- работа с фрагментами текста (копирование, перемещение, удаление и т.п.);
- форматирование текста (установка абзаца, перенос, выравнивание границ строки и т.п.);
- работа с несколькими текстами одновременно посредством многооконного принципа;
- файловая организация работы с текстами и взаимодействие с операционной системой;
- импорт/экспорт текстов из одного формата в другой, в другие прикладные системы;
- работа с разными шрифтами;
- работа со спецсимволами (математические знаки, индексы и т.п.);
- работа с иллюстративным материалом (таблицы, схемы, формулы и пр.);
- проверка правописания;
- поиск и замена фрагментов текста.

Рассмотрим для примера меню популярного в России текстового редактора Лексикон, рис.

2.14.



Рис. 2.14. Главное меню и строка статуса текстового редактора Лексикон

Здесь верхняя строка одновременно является строкой комментариев к пунктам меню, и пользователю легко понять их назначение. Функциональные клавиши имеют достаточно стандартные назначения. Например, F1 - помощь, F9 - переключение с русского на латинский режимы, F10 - выход в меню.

Меню позволяет осуществить настройку редактора на подготовку того или иного документа, связаться с операционной системой. Правила набора, редактирования текста просты и очевидны.

4.3. ИЗДАТЕЛЬСКИЕ СИСТЕМЫ

Общая характеристика

Настольные компьютерные издательские системы приобрели широкую популярность в различных сферах производства, бизнеса, науки, культуры и образования. Издательское дело становится актуальным практически для любой организации. Выпуск информационных бюллетеней, рекламных проспектов, собственных малотиражных газет и даже книг теперь становится необходимым атрибутом информационного обеспечения современных учреждений. Пожалуй, из всех новых информационных технологий, компьютерное издательство является наиболее массовой и практически легко внедряемой.

Настольные издательские системы (desktop publishing) представляют собой комплекс аппаратных и программных средств, предназначенных для компьютерного набора, верстки и издания текстовых и иллюстративных материалов.

Минимальные аппаратно-технические требования для развертывания издательских систем таковы:

- персональный компьютер: IBM PC/AT 486 или другой того же класса, с емкостью оперативной памяти 8 Мбайт;
- дисплей: цветной VGA, SVGA 15/17;
- накопитель на жестком магнитном диске (более 160 Мбайт);
- цветной лазерный принтер;
- цветной струйный принтер;
- цветной сканер,
- мышь.

Существуют различные программные системы, среди них наиболее распространены следующие. Word for Windows, Express Publisher, Illustrator for Windows, Ventura Publisher, PageMaker, TeX. Перечисленные программные системы предназначены для компьютерной верстки. Среди программ подготовки иллюстраций можно выделить следующие: CorelDraw, CorelSystem, Designer, DrawPerfect, GalleryEffect, PC Paintbrush, PhotoStyler, Adobe Photoshop и др

Для издательских систем существуют различные сервисные программы обработки текстовых материалов. Среди них можно выделить 7 основных групп:

- преобразования растровой графики в векторную;
- обработки сканированных изображений;
- обработки шрифтов;
- проверки правописания;
- чтения текстов с помощью сканера;
- русификации программ;
- программы-переводчики.

В начале 1998 г. компания «Corel» анонсировала 8-ю версию графического пакета CorelDraw, которая работает под управлением Windows 95 или Windows NT, включает дополнительные функции, повышающие производительность труда дизайнеров графики и поддерживает работу в Internet. Работа с издательскими системами предусматривает

- работу с меню и с диалоговой панелью,
- работу с текстом и иллюстрациями (набор и формирование);
- редактирование текста и иллюстраций,
- оформление текста (форматирование; выбор шрифтов, гарнитуры, стиля, размеров и т.п.);
- настройку экрана.

Издательские системы имеют стандартные правила работы с меню, командами, сервисными утилитами.

Ниже приведены стандартные наименования клавиш и их комбинаций для выполнения обычных манипуляций при работе с большинством текстовых редакторов и настольных издательских систем.

Программы для DOS И WINDOWS:

Alt или F10	Активизировать командное меню
Alt + буква	Вызов пункта меню (параметра)
Alt + F4	Выход из программы
Alt + Shift + Tab	Переход в окно с предыдущей программой
Alt + Tab	Переход в окно со следующей программой
BkSp	Удалить символ перед курсором
Ctrl + →	Перемещение курсора на слово вправо
Ctrl + ←	Перемещение курсора на слово влево
Ctrl + Del	Удалить документ
Ctrl + End	Перемещение курсора в конец поля ввода
Ctrl + Enter	Начать новую страницу
Ctrl + Home	Перемещение курсора в начало поля ввода
Ctrl + Ins	Копировать в буфер (клипбоард)
Ctrl + N	Создать новый документ
Ctrl + O	Открыть файл (редактировать уже имеющийся)
Ctrl+P	Печать
Ctrl + PgDn	Прокрутка на экран вниз
Ctrl + PgUp	Прокрутка на экран вверх
Ctrl + Q	Выход из программы
Ctrl + S	Сохранить файл
Ctrl + X	Выход из программы
Del	Удалять символ или выделенный фрагмент
Enter	Выполнить действие или команду
Esc	Выйти без сохранения выбранных параметров
Esc	Отменить выполнение
F1	Помощь
F4	Повторить последнюю выполненную команду
Ins	Включить/выключить режим вставки/замены
PgDn	Перемещение курсора на экран вниз
PgUp	Перемещение курсора на экран вверх
Shift + <—	Распространить выделение на символ влево
Shift + —>	Распространить выделение на символ вправо
Shift + Del	Удалить в буфер (клипбоард)
Shift + Ins	Вставить из буфера (клипбоарда)
Shift + Tab	Перемещение курсора в предыдущее поле
Tab	Перемещение курсора в следующую позицию табуляции

Настольная издательская система WORD

В последнее время все большую популярность среди широкого круга пользователей завоевывает текстовый процессор Word для Windows. Прежде всего из-за высококачественных издательских свойств этого продукта фирмы «Microsoft». Несмотря на то, что большинство людей, работающих с WinWord, используют далеко не все его возможности, они могут создавать красиво оформленные, хорошего качества документы, разрабатывать фирменные бланки, визитки, создавать элементы фирменного стиля, рекламные документы. Возможность вставлять таблицы, рисунки, графики, формулы в тексты, создаваемые в Word, также способствует завоеванию абсолютного первенства этого программного продукта.

Word - это многофункциональная программа обработки текстов, настольная издательская система. Ее предназначение:

- набор, редактирование, верстка текста и таблиц;
- управление всеми пунктами меню, опциями и командами с помощью мыши;
- просмотр на дисплее готового к печати документа без затраты бумаги на дополнительную распечатку;
- вставка рисунков и слайдов;

- заготовка бланков, писем и других документов;
- обмен информацией с другими программами;
- проверка орфографии и поиск синонимов.

Для запуска текстового процессора дважды «щелкните» левой клавишей мыши на значке запускаемой программы. Кратко охарактеризуем возникшее интерфейсное окно.

Пиктографическое меню - это строка пиктограмм, состоящая из полей кнопок с изображением той или иной операции на них. В большинстве случаев кнопки дублируют наиболее часто используемые операции, доступные и в обычных меню.

Панель форматирования - это строка пиктограмм, состоящая из элементов, необходимых для оформления текста:

- *полей списков* (они справа снабжены стрелкой, направленной вниз; в результате нажатия мышью на стрелку на экране открывается окно списка, в котором перечисляются доступные для выбора элементы списка);
- *полей пиктограмм* (если фрагмент текста маркирован, то нажатие некоторой кнопки на линейке форматирования применяет связанную с данной кнопкой функцию).

Координатная линейка располагается над окном документа. С помощью координатной линейки можно изменять абзацные отступы, длину строки набора и ширину колонок.

Строка состояния находится на нижней кромке окна Word. В процессе ввода данных в этой строке высвечивается информация о позиции курсора ввода и др.

Линейки прокрутки - расположены вдоль правого и нижнего краев рабочего окна. Перетаскивая мышкой бегунок вы можете быстро передвигаться по тексту.

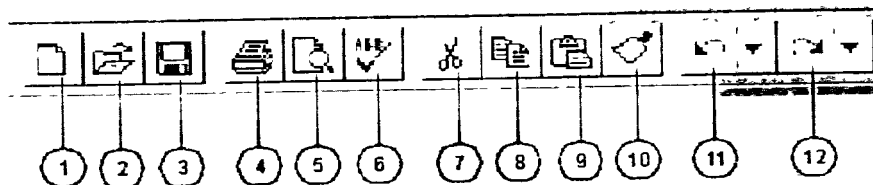


Рис. 2. 15,а. Пиктографическое меню программы Word

На рисунке 2.15, а:

- 1 - создать новый документ;
- 2 - открыть существующий документ;
- 3 - сохранение активного документа;
- 4 - печать активного документа;
- 5 - предварительный просмотр документа перед печатью;
- 6 - проверка орфографии активного документа;
- 7 - удаление выделенного фрагмента в буфер;
- 8 - копирование выделенного фрагмента в буфер;
- 9 - вставка содержимого буфера в позицию курсора;
- 10 - копирование формата;
- 11 - отмена последнего действия;
- 12-повторное выполнение последнего отмененного действия.

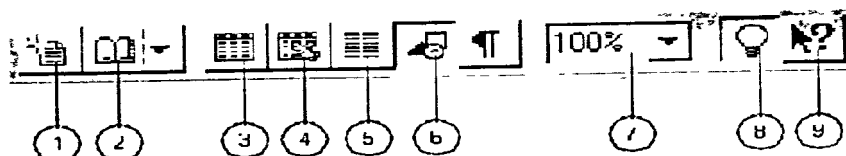


Рис. 2.15,б. Пиктографическое меню программы Word (продолжение)

На рисунке 2.15, б:

- 1 - автоматическое форматирование текста;
- 2 - вставка адреса из адресной книги;
- 3 - вставка таблицы;

- 4 - вставка листа MS Excel;
- 5 - изменение формата колонок;
- 6 - отображение панели рисования;
- 7 - управление масштабом;
- 8 - отображение панели инструментов мастера подсказок;
- 9 - помощь.

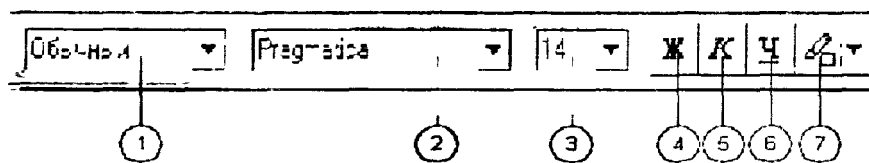


Рис. 2.15.в. Панель форматирования программы Word

На рисунке 2.15, в:

- 1 - применение стиля;
- 2 - изменение шрифта;
- 3 - изменение кегля;
- 4 - установка и отмена полужирного начертания;
- 5 - установка и отмена курсивного начертания;
- 6 - установка и отмена подчёркнутого начертания;
- 7 - выделение цветом выделенного фрагмента.

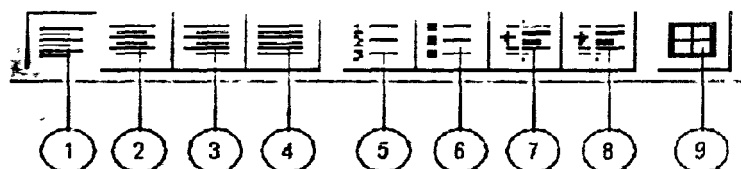


Рис. 2.15,г. Панель форматирования программы Word (продолжение)

На рисунке 2.15, г:

- 1 - выравнивание абзаца по левому краю;
- 2 - выравнивание абзаца по центру;
- 3 - выравнивание абзаца по правому краю;
- 4 - выравнивание абзаца по ширине;
- 5 - создание нумерованного списка;
- 6 - создание маркированного списка;
- 7 - уменьшение отступа или повышение уровня выделенного текста;
- 8 - увеличение отступа или понижение уровня выделенного текста;
- 9 - отображение панели инструментов «Оформление».

Приведенные выше панели используются не только в программе Word; они типичны и для других программ семейства MS Office.

Для выхода из текстового процессора можно дважды «щелкнуть» по соответствующему Word пиктографическому значку системного меню или нажать комбинацию клавиш Alt + F4.

Загрузка документа осуществляется двумя способами:

- 1) нажатием кнопки с изображением раскрытой папки на панели пиктографического меню;
- 2) использованием команды меню *Файл + Открыть*. Далее выбирается нужный файл и нажимается кнопка ОК.

Редактирование текста заключается в удалении, добавлении, копировании и переносе фрагментов текста, проверке орфографии с помощью уже известных клавиш клавиатуры или пиктографического меню. Опишем основные приемы редактирования текста в текстовом процессоре WinWord.

Удаление буквы (символа) - это простейшая функция редактирования. Если вы случайно нажали клавишу с ненужной буквой при вводе текста, то чтобы удалить ошибочно введенный

символ,

- расположенный слева от курсора, надо нажать на клавишу Backspace, при этом курсор передвигается на одну позицию влево;
- расположенный справа от курсора, надо нажать на клавишу Delete, при этом курсор остается на месте

Для удаления участка текста надо его выделить и нажать клавишу Delete.

Как выделить участок текста?

Полоса выделения - это невидимая полоса вдоль левой границы окна документа, используемая для выделения текста с помощью мыши. Чтобы выделить текст надо:

- установить указатель мыши на полосе выделения рядом с текстом;
- нажать левую клавишу мыши и, удерживая ее, передвигать мышь в нужном направлении.

Для отмены удаления

- на панели инструментов пиктографического меню нажать на кнопку с изображением изогнутой влево стрелки;
- или использовать команду меню *Правка + Отменить ввод*.

Добавление текста:

установить текстовый курсор на конец абзаца и нажать клавишу Enter, чтобы начать новый абзац.

Существует два вида **копирования и перемещения участков текста**.

1. Ручная техника.

На небольших расстояниях:

- выделите текст, который хотите переместить или скопировать;
- установите указатель мыши на выделенном тексте, нажмите левую кнопку мыши и удерживайте ее - в этот момент вы как бы «схватили» выделенный участок текста. При этом изменится форма текстового курсора и указателя мыши. Текстовый курсор примет форму штриховой вертикальной линии, а в нижнем конце указателя мыши появится небольшой прямоугольник;
- нажмите клавишу Ctrl, если хотите скопировать выделенный участок текста;
- перемещая указатель мыши, установите штриховой текстовый курсор в ту позицию, куда хотите переместить или скопировать выделенный участок текста, и отпустите кнопку мыши.

2. При переносе текста на большие расстояния воспользуйтесь линейками прокрутки:

- выделите необходимый участок текста;
- прокрутите текст в окне так, чтобы часть, в которую вы хотите перенести или скопировать текст, стала видимой;
- нажмите клавиши Ctrl и Shift и удерживайте их;
- установите указатель мыши в позицию текста, куда хотите вставить свой текст, и нажмите правую кнопку мыши.

Копирование и перемещение участков текста можно также осуществить с помощью Буфера Обмена - участка памяти, в которой временно помещается вырезанный или скопированный текст или графика:

- выделите текст, который хотите переместить или скопировать;
- выберите одну из команд меню *Правка - Вырезать* или *Правка - Копировать*;
- установите текстовый курсор в нужное место;
- выберите команду меню *Правка + Вставить*.

Сохранение текста:

а) на панели пиктографического меню нажать на кнопку с изображением дискеты;

б) или использовать команду меню *Файл + Сохранить*. Откроется диалоговое окно, в котором надо выбрать *Сохранить*.

В поле *Файл* диалогового окна *Сохранить* вы должны ввести имя документа, причем буквы в имени должны быть английскими и расширение у файла .doc. По окончании ввода нажмите на кнопку ОК. Откроется диалоговое окно *Сводка*. В него можно ввести дополнительную информацию о документе, которая будет храниться в том же файле, что и документ. Нажать кнопку ОК.

На экране монитора текст может представляться в различном масштабе и в различном виде, за это отвечает кнопка *Команда* меню *Вид*

Вид + Разметка страницы

Разбивает весь текст на страницы. Используется режим Макета.

Вид + Обычный

Увеличивает страницу на 100%. Изображение текста нормальное.

Вид + Структура Документа

Используется режим Эскиза. WinWord способен отображать на экране непечатаемые символы, такие, как пробел, конец абзаца, табуляцию.

Большинство функций форматирования вызываются с помощью панели инструментов форматирования. Панель инструментов форматирования содержит различные поля, с помощью которых можно вызывать функции редактирования.

Для того чтобы отформатировать символ или группу символов с помощью панели инструментов форматирования, надо

- выделить символ или группу символов;
- открыть список шрифтов и выбрать нужный, задать размер шрифта и выбрать стиль его написания;
- щелкнуть мышью в любом месте документа, чтобы убрать выделение.

Для **вставки рисунков в текст** в текстовом процессоре Word предусмотрена специальная операция в строке меню *Вставка*. Можно вставить в текст стандартные картинки текстового процессора Word:

- установите текстовый курсор в один из пустых абзацев;
- выберите команду меню *Вставка + Рисунок*,
- в появившемся диалоговом окне выберите файл с расширением .wmf (эти файлы находятся в каталоге Clipart);
- если хотите просмотреть содержимое файла прямо в диалоговом окне, установите опцию

Просмотр;

- закончите диалог с помощью кнопки ОК;
- сделайте преобразование рисунка.

Создание собственных рисунков, выполненных в графическом редакторе Paintbrush:

- нарисуйте в графическом редакторе Paintbrush картинку, которую хотите вставить в текст;
- с помощью инструмента *Ножницы* отметьте картинку и скопируйте ее в *Буфер Обмена*;
- с помощью комбинации клавиш Alt + Tab перейдите в текстовый процессор Word (если он у вас не загружен, то перейдите в Диспетчер Программ и запустите его);
- выберите команду меню *Правка + Вставить* или соответствующую кнопку в пиктографическом меню;
- сделайте преобразования рисунка.

Стиль оформления - это набор описания различных форматов абзацев. Каждый формат абзаца имеет свое имя и состоит из последовательности инструкций форматирования. Стиль хранится совместно с документом.

Создание стиля оформления возможно следующим образом:

- выберите команду меню *Формат + Стиль*;
- в диалоговом окне создания стиля в поле *Имя Стиля* введите имя стиля;
- нажмите кнопку *Определить*;
- для определения форматов в группе *Оформление* следует выбрать желаемую кнопку объекта форматирования (*Символ. ., Абзац..., Табуляция..., Обрамление..., Кадр .. или Язык ..*);
- завершение установок в этих диалоговых окнах производится кнопкой ОК, после этого вы опять возвращаетесь в окно создания стиля;
- после создания стиля надо щелкнуть указателем мыши по кнопке *Заменить*;
- в завершение создания стиля надо нажать кнопку *Применить*.

Если у вас уже есть готовый стиль, то для того чтобы применить его к вашему отмеченному тексту, на панели инструментов *форматирования* нужно выбрать первое слева комбинированное окно и выбрать нужный стиль

С помощью текстового процессора Word можно создавать таблицы путем преобразования текста или создания пустой таблицы с последующим заполнением ячеек.

Создание таблицы путем преобразования текста

- введите текст, разделяя будущие ячейки таблицы с помощью табуляции (если текст уже введен, разделите его на ячейки с помощью табуляции);
- выделите текст, который необходимо преобразовать в таблицу;

• включите отображение сетки таблицы с помощью команды меню *Таблица + Линии Сетки*,

- выберите команду меню *Таблица + Преобразовать текст*;
- нажмите кнопку ОК.

Создание пустой таблицы с последующим заполнением ячеек:

1-й способ :

- установите текстовый курсор в том месте, где хотите расположить таблицу;
- выберите команду меню *Таблица + Вставить Таблицу*;
- в появившемся диалоговом окне окажите необходимое число столбцов и строк;

2-й способ:

- установите текстовый курсор в том месте, где хотите расположить таблицу;
- в строке пиктографического меню нажмите на кнопку с изображением таблицы, в появившейся сетке выделите необходимое число строк и столбцов.

Разделительные линии между колонками и строками в таблице при печати на бумаге отсутствуют. На экране вы видите только пунктирные линии, служащие для ориентировки, и на печать не выводимые. Для того чтобы эти линии были видны при выводе на печать надо

- выделить таблицу;
- выбрать команду меню *Формат + Обрамление'*,
- в появившемся диалоговом окне указать тип *Сетка* и задать толщину линии;
- нажать ОК.

В текстовом процессоре Word у пользователя имеется возможность **одновременно открывать вплоть до девяти окон** документов. Независимо от других документов вы можете производить редактирование активного документа.

Окно, находящееся на первом плане, является активным. Для того чтобы перейти в другое окно, надо выбрать команду меню *Окно + Название окна*, к которому хотите перейти.

Для того чтобы закрыть окно, надо выбрать команду меню *Файл + Заккрыть*.

Для перемещения текста из одного окна в другое надо

- загрузить файл с документом, в который хотите переместить текст;
- загрузить файл с документом, из которого хотите переместить текст;
- пометить весь текст, который хотите переместить, и скопировать его в буфер обмена (*Правка + Копировать*);
- перейти в то окно, где загружен документ, в который необходимо переместить текст;
- поставить курсор в то место, в котором должен быть этот текст, и выбрать команду меню *Правка + Вставить*.

Мы описали лишь малую часть возможностей текстового процессора Word, но освоение их поможет производить практически полезную работу с этой сложной программой и понять ее стиль, после чего освоение остальных возможностей не составит труда.

Настольная издательская система PageMaker

PageMaker (PM) - одна из самых мощных и популярных у профессионалов издательских систем. С ее помощью можно подготовить оригинал-макет толстой книги, включающей иллюстрации, формулы, таблицы, другие сложные элементы. Последние версии программы полностью совместимы по интерфейсу с Windows и допускают импорт и конвертацию файлов из любых Windows-приложений; возможен и импорт текстовых файлов. Разработчик современных версий программы PM - фирма «Adobe».

При подготовке к изданию книг PageMaker, предоставляет значительно больше возможностей, чем текстовый процессор Word. Это связано с наличием средств управления проектами, включающими создание шаблонов и стилей, оглавлений и предметных указателей и т.д. К примеру, главы 1 - 6 данной книги подготовлены к изданию в PM.

Для вызова диалоговой панели выбора (установки) параметров работы или выполнения команд необходимо установить курсор мыши на пункт меню или команду и щелкнуть кнопкой или нажать клавиши Alt - F10 и клавишу, соответствующую выделенной букве. Выполнение команд происходит после подтверждения правильности установки всех параметров (выбора значений) активизацией экранной кнопки ОК или нажатием клавиши <Enter>.

Перед началом работы установите принтер и единицу измерения. После этого начните верстку, установив поля и другие параметры, общие для всего издания. Не забудьте выбрать основной шрифт и провести вспомогательные линии.

Изменение масштаба выводимого на экран изображения осуществляется в пункте Page главного меню программы.

Выбрать режим работы с текстом или иллюстрациями можно активизацией соответствующей пиктограммы с помощью мыши или клавиатуры.

Перед началом работы с текстом (иллюстрациями) его необходимо активизировать. Для этого выберите в панели *Инструменты* пиктограмму и щелкните кнопкой мыши по выделенному фрагменту. Можно активизировать одновременно несколько объектов, последовательно выбирая их с помощью мыши при нажатой клавише Shift или отмечая на полосе прямоугольную область, в которой они расположены. Для одновременной активизации всех объектов на полосе нажмите клавиши Ctrl + A или дайте команду *Edit * Select All*.

Для перемещения по тексту издания необходимо переместить указатель мыши в нужное место и щелкнуть левой кнопкой.

Для выделения фрагмента текста выберите соответствующую пиктограмму в панели *Инструменты*. Выделение всего текста издания осуществляется командой *Edit * Select All* или нажатием клавиш Ctrl + A. Выделить фрагмент можно перемещением указателя мыши по тексту с нажатой кнопкой или с помощью клавиатуры.

Помощь:

Help или F1

Выход из программы:

*File * Exit* или Alt + F4

Настольная издательская система TeX

Описанные выше издательские системы плохо приспособлены к набору и подготовке к печати текстов математического характера, с большим числом сложных формул и графиков, специальных математических символов. Для работы с такими материалами, подготовки соответствующих статей и книг американский математик и теоретик программирования Д. Кнут создал издательскую систему TeX.

В отличие от описанных выше систем, TeX не является системой типа «WYSIWYG» - «что набираю, то и вижу на экране». Если набор текста в TeX элементарен, то набор формул и таблиц является, по-существу, написанием программы на специальном макроязыке, что вознаграждается высоким качеством соответствующих текстов.

Контрольные вопросы

1. Какие возможности предоставляют текстовые редакторы?
2. Какие минимальные требования предъявляются к техническим характеристикам компьютера и его периферии для развертывания текстовых редакторов различных уровней?
3. Какие функциональные разделы допускают издательские системы?
4. Подготовьте реферат «История вычислительной техники» с помощью редактора Лексикон; затем, используя правила художественного и технического редактирования, разработайте несколько его вариантов в виде оригинал-макетов с помощью издательской системы Word.

§ 5. СИСТЕМЫ КОМПЬЮТЕРНОЙ ГРАФИКИ

5.1. ПРИНЦИПЫ ФОРМИРОВАНИЯ ИЗОБРАЖЕНИЙ НА ЭКРАНЕ

Существует два способа реализации построения изображений на экране дисплея - **векторный** (функциональный) и **растровый**. В первом случае электронный луч поочередно рисует на экране различные знаки - элементы изображения. На современных персональных компьютерах чаще используется растровый способ изображения графической информации, в котором изображение представлено прямоугольной матрицей точек (**пикселов**), имеющих свой цвет из заданного набора цветов (**палитры**). Графический режим осуществляет видеоадаптер, управляющий

работой электронной трубки и видеопамью, в которой запоминается текущее изображение. Адаптер обеспечивает регулярное отображение видеопамью на экране монитора.

Растровое изображение - это совокупность разноцветных точек. Координаты точек определяются декартовой (прямоугольной) системой с началом координат, как правило, в левом верхнем углу экрана. Абсцисса x точки увеличивается слева направо, ордината y - сверху вниз. Таким образом, любая графическая операция сводится к работе с отдельными точками экрана монитора - пикселями. Существуют специальные графические библиотеки программ, которые предназначены для изображения более сложных объектов, являющихся объединением группы пикселей: наиболее употребимые линии, геометрические фигуры, шрифты и т.п.

Большинство языков программирования имеют свои стандартные графические библиотеки. Так, у Бейсика графические команды являются встроенными; системы программирования Турбо-Паскаль содержат графическую библиотеку (модуль Graph, tru), имеющую в своем составе процедуры и функции обработки простейших графических образов. Тем не менее, многие программисты и разработчики программ предпочитают создавать свои библиотеки графических подпрограмм в соответствии со спецификой своей работы.

В последние годы возрос интерес со стороны пользователей к специальным инструментальным программам машинной графики: графическим редакторам, издательским системам и т.п. В них предоставляется удобный интерфейс для пользователей, автоматизируется большое количество разнообразных действий с графической информацией - от построения простейших рисунков до создания мультипликационных (анимационных) роликов.

Вывод изображения на экран дисплея и разнообразные действия с ним требуют геометрической грамотности, интуиции. Моделирование образной информации на экране дисплея развивает наблюдательность, пространственное воображение, геометрическую интуицию, конструкторские и изобретательские навыки.

5.2. ИЗОБРАЗИТЕЛЬНАЯ ГРАФИКА

При работе с художественной графикой, включая компьютерную графику, необходимо знать следующие понятия.

Композиция - строение, соотношение и взаимное расположение частей, сосредоточие идейно-творческого начала, позволяющего автору произведения искусства целенаправленно организовывать главное и второстепенное и добиваться максимальной выразительности содержания и формы в их образном единстве.

Законы композиции:

1) *закон цельности*: во-первых, наличие конструктивной идеи, объединяющей в единое целое все компоненты произведения; во-вторых, связь и взаимная согласованность всех элементов композиции; в-третьих, неповторимость элементов композиции;

2) *закон контрастов*: контраст света и тени определяет форму предметов; контрасты величин, построения сюжета определяют композицию. Правила композиции:

1) *передача ритма*: ритм в удачной композиции одновременно расчленяет компоненты произведения и объединяет их;

2) *композиционный центр*: выделяется объемом, освещенностью и другими средствами, действующими в соответствии с основными законами композиции.

Мера - характеризует общие принципы строения, целостность предмета, лежит в основе ритма, гармонии, ансамбля в архитектуре.

Симметрия - одинаковость в расположении частей чего-нибудь по противоположным сторонам от точки, прямой или плоскости. Симметрия в композиции создается уравновешенностью ее частей по массам, тону, цвету и форме. Симметрия (геометрическая) - свойство геометрических фигур при котором каждая пара соответственных точек лежит на одном перпендикуляре к данной плоскости по разные стороны и на одинаковом расстоянии от нее.

Пропорция - определенное соотношение частей между собой, соразмерность.

Ритм - равномерное чередование каких-нибудь элементов.

Гармония - соразмерность частей, слияние различных компонентов объекта в единое органичное целое. В истории эстетики гармония - существенная характеристика прекрасного.

Перспектива - искусство изображать на плоскости трехмерное пространство в соответствии с тем кажущимся изменением величины, очертаний, четкости предметов, которое обусловлено степенью удаленности их от точки наблюдения.

При моделировании графических объектов на экране дисплея используют разные методы и способы представления изображений. Можно выделить два принципа моделирования - случайный и детерминированный. На их основе строятся технологии в изобразительной графике, в которой условно можно выделить три направления: художественное, иллюстративное и демонстрационное.

Объектами художественной графики выступают различные узоры, шрифты и другие изображения. При работе с изображениями широко используют простые мотивы и разнообразный геометрический материал. В частности, простые геометрические фигуры в различных сочетаниях и способах размещения (вложения, вращения, симметрии) используются в «живых картинках» для получения муарового эффекта. Удачное сочетание случайного и упорядоченного в любой пропорции с технологиями расположения графических объектов позволяет создавать художественный дизайн.

Самые простые узоры - бордюры - представляют бесконечный ряд равных плоских фигур, расположенных друг за другом таким образом, что элементарная конечная фигура переносится вдоль одного измерения бесконечно. Помимо элементарного мотива для получения бордюра необходимо выбрать группу симметрии бордюра и задать конкретные образующие этой группы из следующего набора движений плоскости:

- параллельный перенос;
- центральная симметрия;
- осевая симметрия;
- скользящая симметрия.

Для бордюров существует четыре абстрактные группы симметрии, которые определяют семь типов симметрии бордюров:

- 1) один параллельный перенос;
- 2) одна скользящая симметрия;
- 3) две осевые симметрии;
- 4) две центральные симметрии;
- 5) одна осевая и одна центральная симметрия;
- 6) один параллельный перенос и одна осевая симметрия;
- 7) три осевые симметрии.

Существует несколько способов построения бордюров по заданному элементарному мотиву и системе образующих ее группы симметрии. Рассмотрим наиболее простые из **них**. Первый способ основан на простом переборе элементов группы - на элементарный мотив действуют поочередно всеми элементами группы симметрии бордюра. Множество полученных образов и будет представлять собой требуемый бордюр. Этот способ неудобен тем, что на каждом этапе необходимо представить очередной элемент группы в виде композиции подходящих степеней образующих.

Второй способ более предпочтителен. На элементарный мотив действуют одним из образующих. Затем на новую фигуру, составленную из исходного мотива и его образа, действуют любым образующим, который не отображает эту фигуру на себя. Получается фрагмент бордюра, состоящий из предыдущей фигуры и ее образа.

Продолжая этот процесс можно построить любую конечную часть бордюра. Преимущество этого способа заключается в том, что на каждом этапе мы будем иметь дело с конкретной симметрией, с нахождением образа конкретной фигуры.

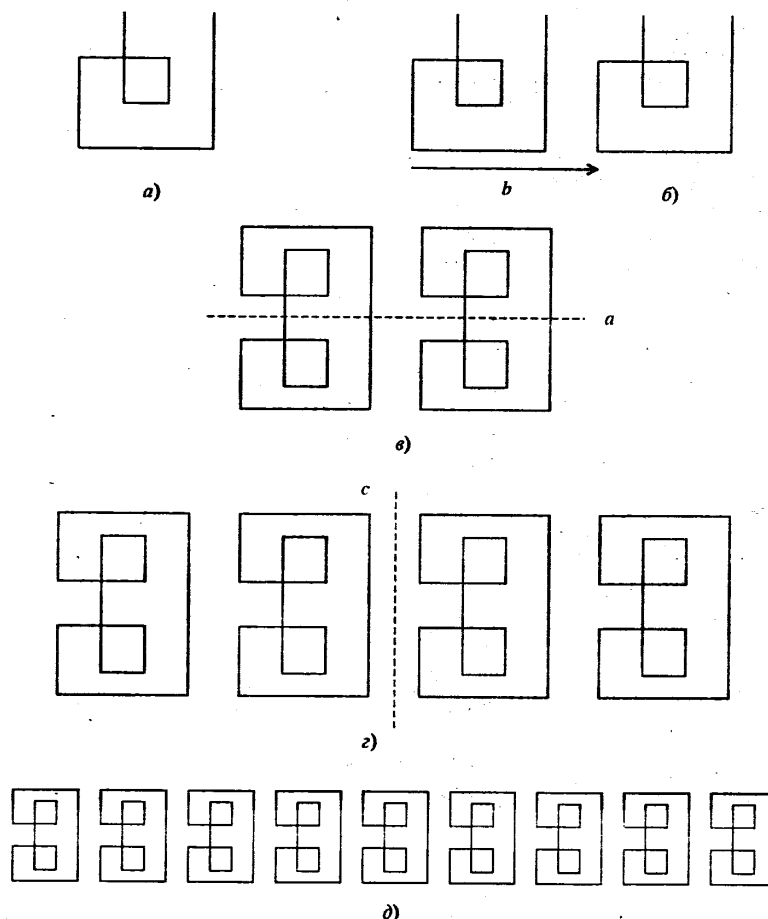


Рис. 2.16. Схема построения бордюра

Рассмотрим, например, процесс построения бордюра с помощью образующих шестого типа симметрии. Создадим элементарный мотив узора (рис. 2.16, а) и определим образующие: параллельный перенос зададим вектором b , (рис. 2.16, б) осевую симметрию прямой a (рис. 2.16, в)

Процесс получения бордюра с помощью заданных образующих вторым из описанных выше способом проиллюстрирован поэтапно. На элементарный мотив действуем параллельным переносом на вектор b , см. схему б. Получим фигуру, состоящую из элементарного мотива и его образа. На новую фигуру действуем осевой симметрией с осью a . Получим фигуру, изображенную на схеме в, на которую действуем параллельным переносом $2b$. В результате получим новую фигуру, см. схему г. Продолжая этот процесс, получаем фрагмент бордюра нужной длины.

Бордюры используются для окантовки обоев, ковров; для настенной росписи, украшающей здания, подземные переходы; в виде металлических решеток для ограждения парков, мостов и садов и пр.

Более зрелищны, привлекательны и интересны по построению орнаменты. Любой орнамент получается переносом узора с помощью двух параллельных переносов, заданных неколлинеарными векторами. Для любого орнамента можно найти сетку, узлы которой составляют вполне определенную систему равных точек орнамента. Различают пять типов плоских решеток: квадратная, прямоугольная, гексагональная, ромбическая и косая. Тип плоской решетки определяет характер переносной симметрии данного орнамента.

В простейшем случае орнамент характеризуется только переносной симметрией. Для построения такого орнамента надо выбрать соответствующую плоскую решетку, заполнить элементарную ячейку решетки определенным рисунком и затем многократно повторить этот рисунок за счет переносов ячейки без изменения ее ориентации. Для построения более сложных по композиции орнаментов рисунок элементарной ячейки заполняется из основного (элементарного) мотива с помощью образующих этого типа симметрии плоских орнаментов. Существует 17 типов симметрии плоских орнаментов, которые определяются следующим образом:

- 1) два параллельных переноса;
- 2) три центральных симметрии;
- 3) две осевые симметрии и параллельный перенос;
- 4) две скользящие симметрии с параллельными осями;

- 5) осевая и скользящая симметрии с параллельными осями;
- 6) симметрия относительно четырех сторон прямоугольника;
- 7) одна осевая и две центральные симметрии;
- 8) две скользящие симметрии с перпендикулярными осями;
- 9) две осевые симметрии с перпендикулярными осями и одна центральная симметрия;
- 10). центральная симметрия и вращение на 90° ;
- 11) симметрия относительно трех сторон прямоугольного равнобедренного треугольника;
- 12) осевая симметрия и вращение на 90° ;
- 13) два вращения на 120° ;
- 14) осевая симметрия и вращение на 120° ;
- 15) симметрия относительно равностороннего треугольника;
- 16) центральная симметрия и вращение на 120° ;
- 17) симметрия относительно трех сторон прямоугольного треугольника с углом 30° .

Несколько слов о частных случаях орнаментов. Бесконечная плоская фигура Φ называется плоским орнаментом, если выполняются следующие условия: 1) среди перемещений, отображающих Φ на себя, существуют неколлинеарные параллельные переносы; 2) среди всех векторов (параллельных переносов), отображающих Φ на себя, существует вектор наименьшей длины. Фигуру называют линейным орнаментом, если плоская фигура отображается сама на себя при параллельных переносах только одного направления (и противоположного ему), причем среди этих переносов существует перенос наименьшей длины.

Большой интерес представляют паркеты. Паркетом называется разбиение плоскости на многоугольники, при котором каждые два многоугольника либо не пересекаются, либо имеют ровно одну общую вершину, либо имеют общую сторону, причем объединение сторон всех многоугольников является плоским орнаментом. Паркет называется правильным, если все многоугольники разбиения правильные (возможно с различным числом сторон) и любую величину паркета можно перевести в любую другую величину некоторым перемещением, отображающим весь паркет на себя.

Интересны орнаменты, заполненные одинаковыми фигурками без промежутков. Фигурку такого орнамента можно создать с помощью геометрических преобразований на основе гексагональной сетки.

Процесс моделирования орнаментов на экране компьютера аналогичен моделированию бордюра: после получения повторяющейся фигурки из элементарного мотива с помощью образующих данного орнамента вывод на экран происходит циклическим изменением координат в двух направлениях. При моделировании орнаментов на основе различных плоских решеток без «заполнения» их элементарной ячейки получают паркеты. Для осуществления компьютерного моделирования графического объекта выбирают подходящее программное инструментальное средство - графический редактор (систему), например, PaintBrush, CorelDraw и т.п., в котором допустимы все возможности, необходимые для описанной работы. Иногда бывает полезным осуществить построение графического образа программированием (на одном из языков программирования) с использованием графических библиотек.

Узор на плоскости получается из элементарного мотива с помощью образующих элементов его группы симметрии. Для бордюра существует 7 различных наборов образующих его групп симметрии, для орнаментов - 17. Чтобы построить узор на компьютере, необходимо

- 1) создать элементарный мотив узора (в графическом редакторе, на языке программирования, сканированием с листа бумаги);
- 2) выбрать и задать образующие;
- 3) построить образы каждой точки элементарного мотива по его образующим, т.е. получить из элементарного мотива изображение конечной повторяющейся фигуры;
- 4) средствами имеющегося компьютера и программного обеспечения размножить получившийся «шаблон» (фигурку), циклически меняя координаты в одном или в двух направлениях.

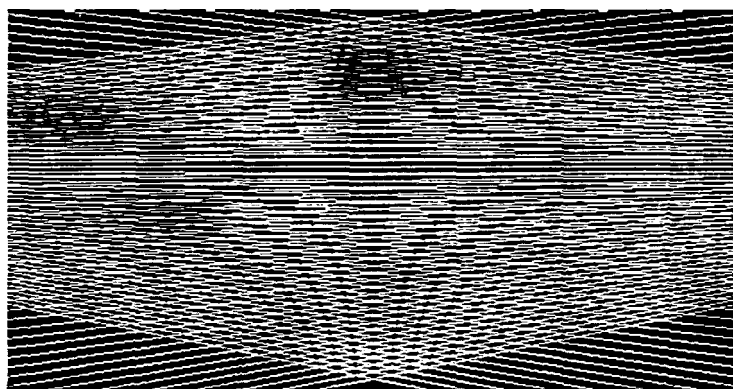
В художественной графике часто обращаются к узорам, получаемым в результате «интерференции» при наложении каких-либо семейств кривых. При пересечении, например, двух семейств прямых под заданным углом получаются «муаровые» эффекты, рис. 2.17.

Моделирование муаровых узоров представляет простой, но занимательный процесс, в котором основная роль принадлежит зрительному восприятию исследователя, который, меняя пара-

метры семейств кривых и их способы пересечения, добивается наилучшей в смысле гармонии красоты узоров.

Используя принцип муаровых узоров, можно моделировать на экране дисплея фейерверки. Простейший одиночный фейерверк можно изобразить по следующему принципу. Из фиксированной точки (x, y) строят отрезки прямых в точку, задаваемую случайным образом и случайным цветом. Возникающие одиночные фейерверки, пересекаясь друг с другом, создают красивые картинки.

Узоры можно моделировать с помощью простых мотивов, в частности, штрихованного угла. Штрихованный угол задается координатами трех точек и изображается семейством отрезков, соединяющих точки разбиения сторон угла с одинаковыми номерами. Подобные мотивы удобно оформлять в виде подпрограммы, а затем использовать при моделировании различных изображений.



. Рис. 2.17. Муаровый рор

Сочетание случайного и упорядоченного в любой пропорции при изображении графических объектов дает эффект. Например, на основе простого мотива - угла, вершина которого находится на горизонтальной прямой, - можно изобразить «лес». При упорядоченном размещении «деревьев» имеем «посадку».

Подобные технологии широко используют в иллюстративной графике. В настоящее время **иллюстративная графика**, в первую очередь, связана с изображением графического материала в издательских системах.

Иллюстративный материал - схемы, эскизы, географические карты, чертежи и др. - можно создавать различными графическими редакторами, системами. Здесь важна легкость и быстрота формирования и преобразования графических изображений для тех или иных приложений. В последнее время большой интерес вызывает иллюстративный материал, представленный в демонстрационной, динамической форме.

Демонстрационная графика связана с динамическими объектами. В технологии изображения динамических объектов используют три основных способа: рисование -стирание, смену кадров (страниц), динамические образы.

Достаточно просто организовать перемещение фрагмента рисунка на экране. Для этого надо

- создать этот фрагмент в нужном месте экрана;
- стереть фрагмент, рисуя его цветом фона или используя процедуру очистки экрана;
- снова нарисовать фрагмент в другом месте экрана, и так далее.

Еще один способ организации движения объектов на экране, широко применяющийся в компьютерных играх, связан с использованием нескольких экранных страниц. В любой момент времени одну из страниц можно сделать видимой и посмотреть ее содержимое на экране. Визуальная страница обычно пассивна, т.е. на ней нельзя выполнять графические процедуры. Другую страницу можно объявить активной. Активная страница невидима для пользователя, но на ней можно подготовить другой рисунок. Меняя страницы ролями, можно создать мультипликационный сюжет

В обучающих программах и в компьютерных играх часто используют динамические образы (в том числе спрайты). При этом используют динамическую память для хранения прямоугольных

фрагментов рисунков и последующего вывода их в нужное место экрана.

5.3. ГРАФИЧЕСКИЕ РЕДАКТОРЫ

Создать объекты иллюстративной графики, включая динамические ролики, можно средствами программирования, а также графическими редакторами и системами. Рассмотрим два из них.

Редактор Paint

Графический редактор Paint, входящий в комплект стандартных программ MS Windows 95, позволяет, используя манипулятор «мышь», выполнять черно-белые и цветные рисунки, обрамлять их текстом, выводить на печать. В Paint можно работать с фрагментами графических изображений: копировать, перемещать, поворачивать, изменять размеры, записывать на диск и считывать с диска. С помощью Paint можно обрабатывать графические изображения, а также считывать и записывать в файл полностью или частично изображение с дисплея, если монитор работает в графическом режиме.

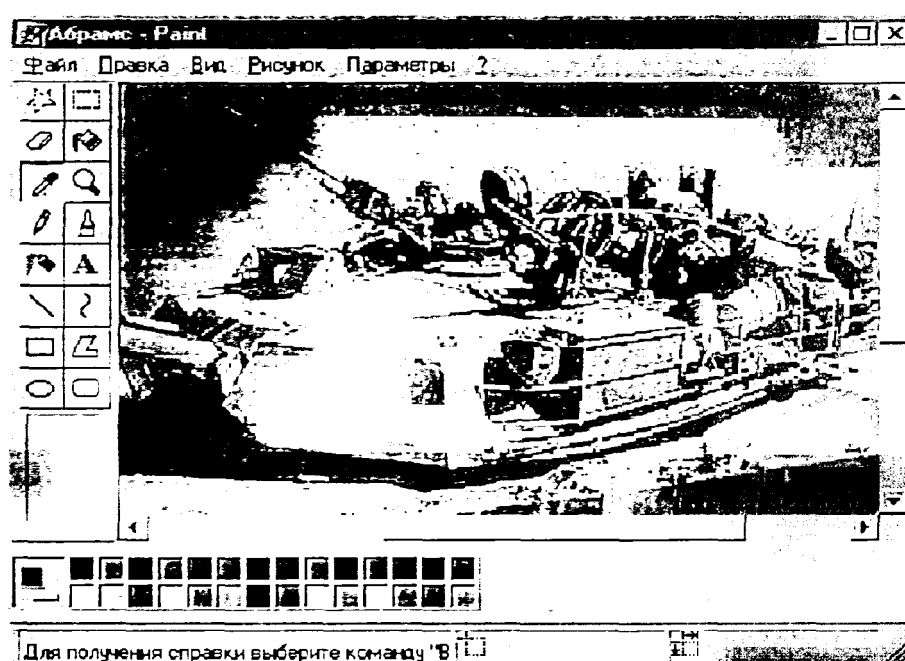


Рис. 2.18. Рабочее окно программы Paint

После загрузки пакета появляется рабочий экран редактора (рис. 2.18). Большую часть экрана занимает рабочее поле, окрашенное в фоновый цвет. Над рабочим полем - меню, позволяющее выполнять команды редактирования. Слева от рабочего поля расположена панель инструментов, на которой высвечен инструмент, в данный момент являющийся рабочим. Под рабочим полем находится палитра. С левого края палитры показаны два вложенных квадрата, внутренний из которых окрашен в рабочий цвет, а внешний - в фоновый. В левом нижнем углу экрана выводится калибровочная шкала, которая позволяет устанавливать ширину рабочего инструмента (кисти, резинки и т.д.). Установленная в данный момент ширина инструмента отмечена стрелкой. Вдоль нижнего правого и правого края рабочего поля находятся линейки прокрутки для перемещения рабочего поля по картинке, если размеры картинки больше размеров рабочего поля.

Общие правила работы с редактором таковы. Для выбора (установки) параметров работы и выполнения команд в Paintbrush необходимо поместить указатель мыши на пункт меню и щелкнуть левой кнопкой. Выход из программы: *File/Quit*.

Программа *Freeze* предназначена для сохранения выводимого на экран изображения в графическом РСХ-файле с одновременным сохранением оформления экрана, возможностью последующего редактирования данного изображения и вставки его в качестве иллюстрации в текстовые редакторы и настольные издательские системы. Установка программы происходит автоматически

при запуске Paintbrush.

Редактор CorelDraw

Начиная работать с графическим редактором CorelDraw, мы прежде всего видим его рабочее окно, рис. 2.19.

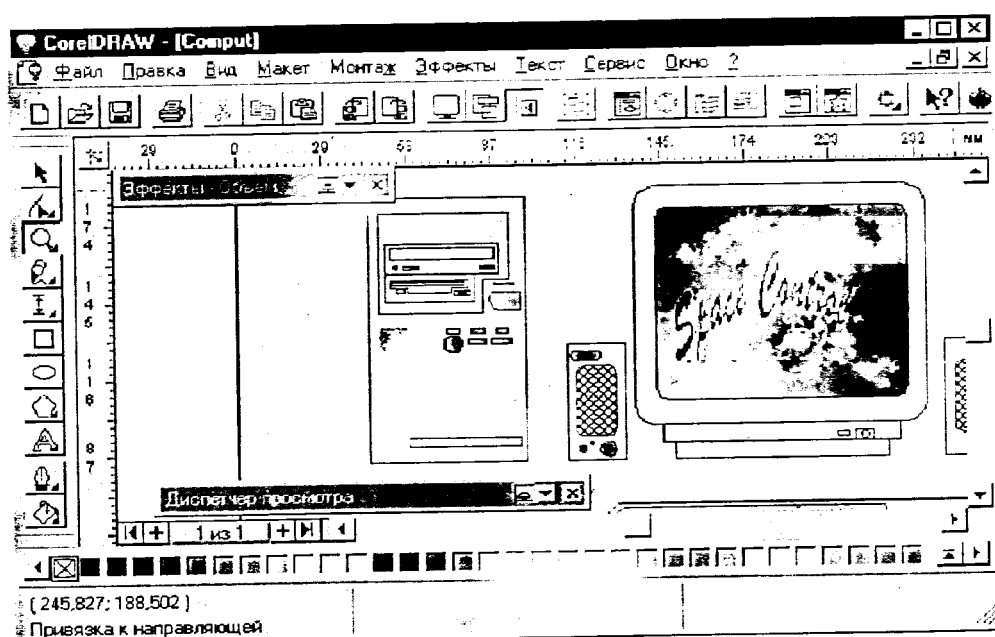


Рис. 2.19. Рабочее окно программы CorelDraw

Чтобы активизировать **меню выбора** (установки) **параметров** работы или выполнения команд, установите указатель мыши на пункт меню и щелкните левой кнопкой или нажмите клавишу Alt и клавишу, соответствующую выделенной букве. В некоторых случаях для облегчения выполнения наиболее часто употребляемых команд ввод их с клавиатуры осуществляется нажатием комбинации клавиш или определенной клавиши. Выполнение команд происходит после подтверждения правильности установки всех параметров или выбора значений активизацией экранной кнопки ОК в диалоговой панели или нажатием клавиши Enter.

Выход из меню - переместить указатель мыши за пределы меню и щелкнуть кнопкой или нажать клавишу Esc.

Выход из программы: *File/Quit* или Alt + F4.

Для вызова **диалоговой панели** выбора (установки) параметров работы или выполнения команд необходимо установить указатель мыши на пункт меню или команду и щелкнуть кнопкой или нажать клавишу Alt и клавишу, соответствующую выделенной букве. Выполнение команд происходит после подтверждения правильности установки всех параметров (выбора значений) активизацией экранной кнопки ОК или нажатием клавиши Enter.

Для **редактирования рисунка** следует активизировать пиктограмму с помощью мыши или нажатием клавиши *Пробел*, переместить указатель на любую точку контура рисунка и щелкнуть кнопкой. Выбранный рисунок будет окружен восемью квадратами черного цвета. Можно выделить одновременно несколько объектов, последовательно выбирая их с помощью мыши при нажатой клавише Shift или отмечая на экране прямоугольную область, в которой они расположены. Для одновременного выбора всех рисунков на экране необходимо активизировать пиктограмму, переместить указатель мыши в один из углов выбираемого прямоугольного контура, нажать кнопку и, не отпуская ее, переместить указатель в противоположный угол и отпустить кнопку. Контур будет изображен штриховой линией. Для отмены выбора - переместить указатель мыши за контур и щелкнуть кнопкой.

Для изменения масштаба выводимого на экран рисунка необходимо активизировать соответствующую пиктограмму.

Работа с текстом начинается с активизации пиктограммы текста. Для перемещения по тексту в диалоговой панели *Text* необходимо переместить указатель мыши и щелкнуть левой кноп-

кой. Однако в программе для перемещения предусмотрены также специальные клавиши.

Для выделения фрагмента текста необходимо активизировать пиктограмму с помощью мыши или клавиши *Пробел*. Переместите указатель мыши в один из углов выбираемого прямоугольного контура, нажмите кнопку *К*, не отпуская ее, переместите указатель в противоположный угол, отпустите кнопку. Весь контур будет окружен штриховой линией. Для выделения отдельного символа необходимо активизировать пиктограмму, переместить указатель мыши в один из углов выбираемого прямоугольного контура, нажать кнопку *и*, не отпуская ее, переместить указатель в противоположный угол. Выбранный объект не будет окружен рамкой из квадратов, однако на нем будут выделены все узловые точки. Выбор нескольких символов осуществляется аналогично при нажатой клавише *Shift*.

Прежде чем начать оформление ранее введенного текста, его необходимо выделить одним из выше описанных способов.

5.4. ДЕЛОВАЯ ГРАФИКА

Одним из первых приложений компьютерной графики стало отображение данных экономических расчетов.

Графические представления расчетных и статистических данных удобно представлять в виде схем, диаграмм, гистограмм и графиков. Различают следующие их виды:

гистограмма - группа столбцов, пропорциональных по высоте определенным числовым значениям;

круговая диаграмма - секторы круга, углы которых пропорциональны элементам данных;

линейный график - отображение исходных величин в виде точек, соединенных отрезками прямых линий;

временная диаграмма - последовательность операций или процессов определенной длительности (измерение динамических процессов);

структурная схема - представление сложных объектов в виде дерева или графа;

круговая гистограмма - представление относительных величин объектов, которым на изображении сопоставляются размеры и расположение кругов в прямоугольной системе координат.

Из числа средств прикладного программного обеспечения общего назначения графическое представление данных лучше всего развито в электронных таблицах и в СУБД.

Одним из первых практических применений машинной графики было автоматическое построение графиков функции в различных системах координат. Обычно графики функций строят в декартовых координатах (в прямоугольной системе, рис. 2.20).

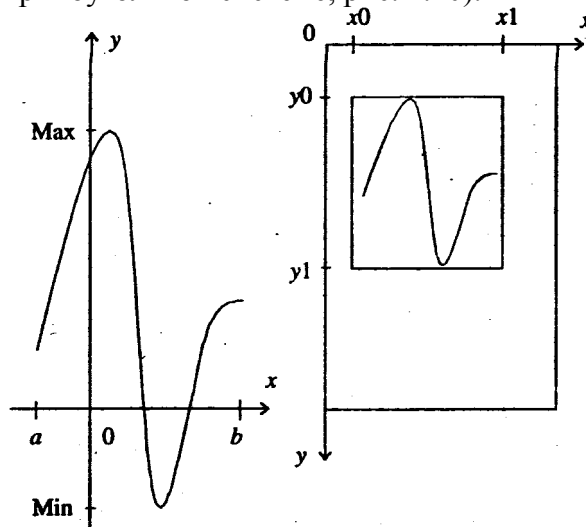


Рис. 2.20. Построение на экране графиков функций (в декартовой системе координат)

В общем виде алгоритм построения графика заданной функции $y = f(x)$ на отрезке $[a, b]$ заключается в следующем.

1. Определяем область значений функции, для чего найдем максимальное по модулю зна-

чение функции на заданном отрезке $[a, b]$, $m = \max(\text{abs}(f(x)))$ для всех x из $[a, b]$.

Примем для удобства, что минимальное значение функции на отрезке совпадает с максимальным, но с обратным знаком. Таким образом, область значений функции лежит в интервале $[-m, m]$.

Поиск максимума можно осуществить разными способами, например, табулируя функцию $f(x)$ на отрезке с разбиением на n частей и определяя максимальное значение в массиве чисел $Y_i = f(x_i)$, где $x_i = a + i \cdot (b - a) / n$, для $i = 0, \dots, n$.

2. Задаем координаты окна x_0, y_0, x_1, y_1 на графическом дисплее, в котором будем строить график функции.

3. Формулы преобразования координат x, y точек прямоугольника $[a, b] \cdot [-m, m]$ обычной декартовой системы в соответствующие координаты u, v окна $[x_0, x_1] \cdot [y_0, y_1]$ графического экрана можно задать в следующем виде :

$$u = x_0 + (x - a)(x_1 - x_0)/(b - a),$$
$$v = (y_0 + y_1) / 2 - f(x) (y_1 - y_0) / (2m).$$

Тогда автоматическое построение графика функции на экране дисплея осуществляется путем установки точек (u_i, v_i) , соответствующих точкам $(x_i, f(x_i))$, выбранным в декартовой системе. Часто бывает полезно соединять полученные точки отрезками или специальными линиями, что программы могут делать (или не делать) по желанию пользователя.

4. Далее можно оформить график, нарисовав оси координат, нанести масштабные сетки, вывести соответствующие обозначения и комментарии. Оси координат на графическом экране в заданном окне легко построить, вычислив экранные координаты начала выбранной декартовой системы $\{xv, yv\}$:

$$xv = x_0 - a(x_1 - x_0)/(b - a),$$
$$yv = (y_0 + y_1) / 2.$$

5.5. ИНЖЕНЕРНАЯ ГРАФИКА

Компьютеризацию чертежных и конструкторских работ проводят давно и в настоящее время используют различные системы автоматизации проектных работ (САПР). Аббревиатуру САПР впервые использовал основоположник этого научного направления Айвен Сазерленд в своих лекциях, прочитанных в Массачусетском технологическом институте в начале 60-х годов. Фактически инженеры применяли компьютеры для решения сложных задач проектирования еще в эпоху ранних послевоенных моделей универсальных компьютеров, а первые образцы специализированного оборудования САПР были созданы уже в середине 50-х годов. Однако широкое распространение САПР обусловлено появлением микропроцессорной техники, предоставившей возможности создавать, модифицировать и обрабатывать сложные графические изображения на экране монитора.

В настоящее время САПР обозначает аппаратно-программный комплекс, поддерживающий процесс проектирования с использованием специальных средств машинной графики, поддерживаемых пакетами программного обеспечения, для решения задач, связанных с проектной деятельностью. В совокупности развитая САПР представляет собой специализированную информационную систему. Сфера применения САПР охватывает такие разные области приложения, как архитектура, гражданское строительство, картография, медицина, геофизика, разработка моделей одежды, издательское дело, реклама.

Полная система САПР состоит из компонентов аппаратного и программного обеспечения. Общими компонентами аппаратного обеспечения системы САПР являются ЦП (центральный процессор), несколько рабочих станций, разделяемая между рабочими станциями периферия.

Состав типичной системы САПР:

дисплей (графический и алфавитно-цифровой);

процессор;

клавиатура;

устройство управления курсором (мышь, дигитайзер);

электронный командный планшет;
принтер.

Одним из наиболее давних и популярных средств автоматизированного проектирования является система АВТОКАД (AutoCad). АВТОКАД не является проблемно-ориентированной системой, т.е. не содержит специализированных баз данных, экспертных систем и многого из того, что входит в состав специализированной интеллектуальной САПР. АВТОКАД - достаточно простая универсальная система. Ее возможности таковы:

- развитая система экранных меню;
- высокая точность графической информации;
- разбивка информации (расслоение);
- прочерчивание на дисплее координатной сетки;
- средство захвата графических объектов;
- мощное редактирование;
- отображение параметров графических характеристик;
- полуавтоматическая и автоматическая проstanовка размеров;
- штриховка;
- работа с блоками.

После запуска АВТОКАДа на текстовом экране появляется главное меню:

О - выход;
1 - создание нового чертежа;
2-редактирование существующего чертежа;
3 - вывод на плоттер (графопостроитель);
4 - вывод на принтер;
5-конфигурация;
6 - файловые утилиты;
7-шрифты;
8 -стыковка со старыми версиями.

Режимы экранного меню:

AUTOCAD	- выход в головное меню;
* * * *	- режим объектного захвата;
BLOCKS	-работа с блоками;
DISPLAY	- работа с изображением без его изменения;
SETTINGS	-настройка;
DIM	- обезразмеривание;
EDIT	-редактирование;
DRAW	-рисование;
LAYER	-работа со слоями;
INQUIRY	-справки о примитивах;
UTILTTS	- выход в ДОО, запись чертежей в разных форматах;
PLOT	-получение твердой копии и т.д.

В режиме DRAW (рисуи) имеется возможность строить графические примитивы и проводить с их помощью синтез изображений. Например, здесь существует восемь способов рисования дуг:

- по трем точкам на дуге /3 points/;
- по начальной точке, центру и длине хорды /S, C, L/;
- по начальной точке, центру и заключенному углу /S, C, A/;
- по начальной точке, конечной точке и радиусу /S, E, R/;
- по начальной точке, конечной точке и заключенному углу /S, E, A/;
- по начальной точке, конечной точке и исходному направлению /S, E, D/;
- по продолжению предыдущей линии или дуги /CONTIN:/.

Можно выделить следующие правила изображения дуги:

- 1) обычно дуга строится против часовой стрелки от точки к точке;
- 2) если есть в выбранной опции возможность задать угол, то отрицательный угол позволяет рисовать дугу по часовой стрелке;
- 3) по умолчанию дуга рисуется по начальной, промежуточной и конечной точкам.

При выборе в основном меню АВТОКАД режима *Edit* имеется возможность стирать созданные объекты, возвращать случайно стерты, перемещать и копировать, вычерчивать сопряжения между двумя существующими объектами, снимать фаски, разбивать объект на части, зеркально отображать, поворачивать, увеличивать и уменьшать, отсекасть и т.п.

Практически все команды редактирования запрашивают выбор одного или нескольких объектов для обработки. Совокупность таких объектов называется набором выбора. Когда АВТОКАДу требуется не один объект, а набор выбора, появляется подсказка *Select objects:* (выбрать объекты) и на экране вы видите маленький прямоугольник - мишень для выбора объектов. Способы выбора объектов указываются в опциях соответствующей команды.

5.6. НАУЧНАЯ ГРАФИКА

Компьютерная графика представляет значительный интерес для научных исследований. В частности, она выступает как средство формирования научной документации с использованием специальной нотации - математических знаков, индексов, шрифтов и т.п. В последнее время ученые чаще стали обращаться к имитационному моделированию на компьютере.

В компьютерной графике большое значение имеют методы и способы **геометрического моделирования**. Модели, геометрические преобразования составляют в настоящее время основу теории компьютерной графики и геометрического моделирования. **Аналитические модели** - это набор чисел, логических параметров, играющих роль коэффициентов в уравнениях, которые задают графический объект заданной формы. Например, аналитической моделью окружности на плоскости в параметрической форме являются уравнения

$$\begin{aligned}x &= x_0 + R \cdot \cos A, \\ y &= y_0 + R \cdot \sin A,\end{aligned}$$

где x_0, y_0 - координаты центра, R - радиус, A - угол. Параметрическое задание образов широко применяется в машинной графике и геометрии. Изображение окружности можно осуществить установкой последовательных точек (близко расположенных), изменяя генерирующий параметр A от 0 до 360° .

Координатные модели - это массивы координат точек, принадлежащих объектам. Например, поверхность задается массивом точек $Z = f(x, y)$ на координатной сетке $[x_i, y_j]$. Если точки в модели расположены в том же порядке, что и на линии образа, то модели называют упорядоченными. Помимо координат, в модели могут быть указаны дополнительные характеристики проекции касательных или нормальных векторов.

Приближенные модели содержат аппроксимации кривых методами вычислительной геометрии. Например, изображение гладких кривых можно осуществить ломаными линиями: линейными, параболическими или сплайнами. Используя вышеперечисленные геометрические модели, можно создавать различные демонстрационные картины. Например, модель Солнечной системы для наглядности удобно представить в динамической форме. Организуем движение точки (Земли) по окружности, в центре которой размещается круг (имитация Солнца). Установку точки на орбите осуществим по параметрическим формулам окружности:

$$\begin{aligned}X_0 &= 320 + r_1 \cdot \sin(A_1); \\ Y_0 &= 240 + r_1 \cdot \cos(A_1),\end{aligned}$$

где r_1 - радиус орбиты Земли, A_1 - параметрический угол, меняющийся от 0 до 360° . Чтобы организовать движение, достаточно в цикле устанавливать точку с координатами (x_0, y_0) для всех углов A_1 , принимающих значения от 0 до 360° с шагом h . Аналогичная процедура справедлива и для второй точки (Луны), которая изображается по подобным формулам, в которых центр орбиты

(Земля) является подвижным:

$$x = x_0 + r \cdot \sin(A);$$
$$y = y_0 + r \cdot \cos(A),$$

где r - радиус орбиты Луны, A - угол вращения.

Контрольные вопросы и упражнения

1. Составьте каким-либо средством машинной графики бордюры каждого типа симметрии из следующих элементарных мотивов, рис. 2.21.

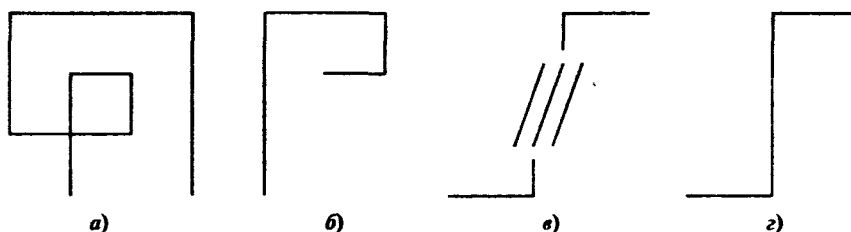


Рис. 2.21.

2. Постройте орнаменты различного типа симметрии из выбранного произвольного элементарного мотива каким-либо средством машинной графики.

3. Задана высота (м) над уровнем моря вершин: Мак-Кинли - 6200, Логан - 6100, Элберт - 440, Робсон - 4000, Митгелл - 2000. Составьте по этим данным столбчатую диаграмму.

4. По данным упражнения 3 составьте круговую диаграмму.

5. Постройте график функции $y = x \sin(1/x)$.

6. Создайте мультипликацию: вращение электрона в модели атома.

7. Изобразите шестиугольную призму.

8. Подготовьте иллюстрацию ко Дню учителя.

9. Организуйте в школе (вузе) компьютерный вернисаж.

§ 6. БАЗЫ ДАННЫХ И СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

6.1. ПОНЯТИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Базы данных - важная составная часть **информационных систем**. Здесь мы ограничимся лишь кратчайшими общими сведениями об информационных системах, сосредоточив внимание на базах данных как таковых.

Информационные системы предназначены для хранения и обработки больших объемов информации. Изначально такие системы существовали в письменном виде. Для этого использовались различные картотеки, папки, журналы, библиотечные каталоги и т.д. Любая информационная система должна выполнять три основные функции: ввод данных, запросы по данным, составление отчетов.

Ввод данных. Система должна предоставлять возможность накопления и упорядочивания данных. Необходимо обеспечить просмотр этих данных, внесение в них изменений и дополнений с тем, чтобы поддерживать актуальность информации.

Запросы по данным. В системе должна существовать возможность находить и просматривать отдельные части накопленной информации.

Составление отчетов. Время от времени возникает необходимость обобщать и анализировать большую группу данных (или даже все данные) информационной системы, представляя ее в виде документа.

Обслуживание информационных систем, реализованных в письменном (бумажном) виде, сопряжено со многими трудностями: чем больше информационная система, тем больше бумаги (карточек) и места требуется для их хранения (в этом можно убедиться на примере библиотеки);

много времени тратится на поиск нужной информации. Сложности возникают при обновлении, анализе и обработке данных.

Предположим мы хотим собрать информацию про альбомы музыкальных групп. Пусть имеется информация о некоторых альбомах: 1965, Led Zeppelin 4, Lp, Help!, Atlantic, 1971. Lp(England), EMI. 1970, Flash Gordon, Parlophone, 1980, Led Zeppelin 3, Soundtrack, Lp, Atlantic. Этот список мало о чем говорит. Извлечь какую-либо информацию из этого набора данных практически невозможно.

Представим данные в виде табл. 2.2.

Таблица 2.2 Информация об альбомах музыкальных групп

Название альбома	Год выпуска	Тип	Фирма альбома
Help!	1965	Lp (England)	Parlophone
Led Zeppelin 4	1971	Lp	Atlantic
Led Zeppelin 3	1970	Lp	Atlantic
Flash Gordon	1980	Soundtrack	EMI

Теперь воспринимать и использовать информацию стало гораздо удобнее. Представленная таблица является **информационной моделью**. Объектами, отраженными в этой модели, являются музыкальные альбомы (групп), причем все данные взаимосвязаны.

6.2. ВИДЫ СТРУКТУР ДАННЫХ

В информатике совокупность взаимосвязанных данных называется **информационной структурой**, или **структурой данных**. В нашем примере объектами модели являются музыкальные альбомы. Свойства же этих объектов находятся в столбцах таблицы («Название альбома», «Год выпуска», «Тип альбома», «Фирма»), их называют **атрибутами** объектов. Таким образом, каждая строка таблицы - есть совокупность атрибутов объекта. Такую строку называют **записью**, а столбец - **полем записи**.

Помимо сведений, указанных в атрибутах, табличная организация данных позволяет получить дополнительную информацию. К примеру, нетрудно узнать (в предположении, что наша табл. 2.2 заполнена данными):

- какая группа выпустила больше альбомов за определенный период;
- число альбомов данной группы;
- сколько имеется альбомов типа Soundtrack (музыка к фильму);
- какая фирма выпустила наибольшее число альбомов данной группы.

Табличная организация данных называется также **реляционной**. Кроме табличной структуры данных существуют другие виды структурной организации данных.

Для **иерархических структур** (рис.2.22) характерна подчиненность объектов нижнего уровня объектам верхнего уровня. Важно отметить, что в дереве, между верхними и нижними объектами, задано отношение «один ко многим» (т.е. одной группе соответствует много альбомов, одному альбому соответствует много песен).

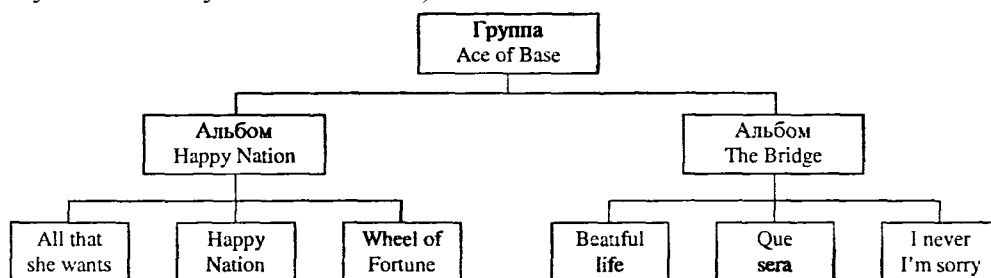


Рис. 2.22. Пример иерархической организации данных

Несмотря на то, что в атрибутах, описывающих песню, нет названия альбома, глядя на дерево по линиям связи можно сказать, какая песня принадлежит альбому. Благодаря линиям связи можно определить принадлежность альбома группе. Из данной иерархической структуры можно

узнать:

- в каком альбоме больше песен;
- число альбомов выпущенных группой;
- есть ли в альбомах одинаковые песни и т.д.

Сетевую структуру данных можно представить в виде схемы, рис. 2.23.

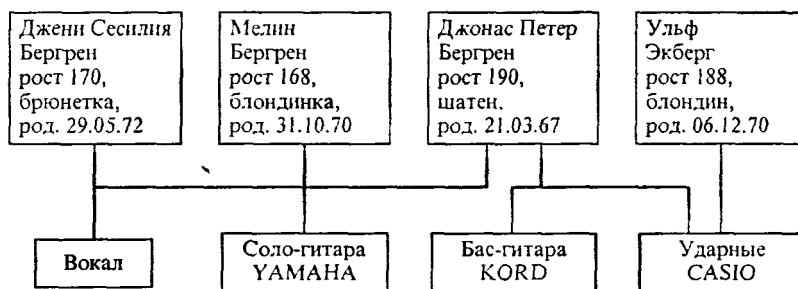


Рис. 2.23. Пример сетевой организации данных

Глядя на рис. 2.23, можно определить, какими инструментами владеет музыкант, является ли он вокалистом. В этом случае есть два уровня взаимосвязанных объектов, но отношение между ними «многие ко многим».

Пусть в этой сетевой структуре данные о музыкантах и «инструментах» состоят из следующих атрибутов: музыкант - ФИО, рост, цвет волос, время рождения; инструмент - название инструмента, какой фирмой изготовлен инструмент.

Тогда схема позволяет ответить на следующие вопросы:

- гитары какой фирмы предпочитает большинство музыкантов;
- какой музыкант владеет наибольшим количеством инструментов и др.

Построение структуры данных происходит в следующем порядке:

- определяются объекты описания;
- определяются структуры этих объектов;
- выбирается тип структуры, отображающий отношения между объектами (табличная, иерархическая, сети);
- строится конкретная информационная структура.

6.3. ВИДЫ БАЗ ДАННЫХ

Дадим основное определение. База данных - это реализованная с помощью компьютера информационная структура (модель), отражающая состояние объектов и их отношения.

Следует учесть, что это определение не является единственно возможным. Информатика в отношении определений чаще всего не похожа на математику с ее полной однозначностью. Если подойти к понятию «база данных» с чисто пользовательской точки зрения, то возникает другое определение: база данных - совокупность хранимых операционных данных некоторого предприятия. Все дело в том, какой аспект доминирует в рассмотрении; в данной главе первое из определений более уместно.

Поскольку основу любой базы данных составляет информационная структура, базы данных делят на три рассмотренные выше типа: табличные (реляционные), сетевые, иерархические.

Опыт использования баз данных позволяет выделить общий набор их рабочих характеристик:

- **полнота** - чем полнее база данных, тем вероятнее, что она содержит нужную информацию (однако, не должно быть избыточной информации);
- **правильная организация** - чем лучше структурирована база данных, тем легче в ней найти необходимые сведения;
- **актуальность** - любая база данных может быть точной и полной, если она постоянно обновляется, т.е. необходимо, чтобы база данных в каждый момент времени полностью соответствовала состоянию отображаемого ею объекта;
- **удобство для использования** - база данных должна быть проста и удобна в использовании и иметь развитые методы доступа к любой части информации.

Соответственно возможностям организации реляционных, иерархических и сетевых ин-

формационных структур, существуют и аналогичные виды баз данных. В них данные представлены в формах, адекватных соответствующим структурам. Однако иерархические и сетевые базы данных являются гораздо менее распространенными, чем реляционные и не могут быть реализованы с помощью наиболее популярных СУБД, входящих в состав программного обеспечения ЭВМ, поэтому на них далее останавливаться не будем.

Реляционные базы данных

Наиболее распространенными в практике являются реляционные базы данных. Название «реляционная» (в переводе с английского relation - отношение) связано с тем, что каждая запись в таблице содержит информацию, относящуюся только к одному конкретному объекту.

Всякое отношение должно иметь свое имя. Пусть есть отношение с названием «Альбомы группы». В этом случае структура базы данных, состоящая из одной таблицы, запишется так: *Альбомы группы (название альбома, год выпуска, тип альбома, фирма)*. Однако чаще база данных строится на основе нескольких таблиц, связанных между собой через общие атрибуты. Пусть, например, в базе данных «Рок-энциклопедия» содержатся две таблицы - 2.3, а и 2.3, б.

Таблица 2.3, а Музыкальные альбомы групп

Код альбома	Код группы	Название альбома	Год выпуска	Тип альбома	Фирма
25	1	Help!	1965	Lp (English)	Pariophone
36	2	Led Zeppelin 4	1971	Lp	Atlantic
35	2	Led Zeppelin 4 -	1970	Lp	Atlantic
34	3	Flash Gordon	1980	Soundtrack	EMI

Таблица 2.3, б Рок группы

Код группы	Название группы	Страна	Дата создания	Дата распада
1	The Bealles	Англия	1963	1970
2	Led Zeppelin 4	Англия	1989	-
3	Flash Gordon	Англия	1991	-

Эти две таблицы связаны между собой общим полем «Код группы». Поле «Код альбома» в таблице 2.3, а создается для того, чтобы отличать альбомы друг от друга. Это очень важно, так как в таблице могут находиться альбомы с одинаковыми названиями.

Необходимость использования больше одной таблицы станет заметной, если объединить эти таблицы в одну (табл. 2.4).

Таблица 2.4 Объединение таблиц 23

Название группы	Страна	Дата создания	Дата распада	Название альбома	Год выпуска	Тип альбома	Фирма
The Beatles	Англия	1963	1970	With the Beatles	1963	Lp	Pariophone
The Beatles	Англия	1963	1970	Please, please me	1963	Lp	Pariophone
The Beatles	Англия	1963	1970	Rubber soul	1963	Lp	Pariophone

Из таблицы 2.4 видно, что при внесении в нее данных об альбомах определенной группы каждый раз приходится дублировать информацию первых четырех полей таблицы. Многократное сохранение в БД одних и тех же данных (название группы, страна, дата создания, дата распада) приведет к неэффективному использованию памяти, к тому же существенно возрастет вероятность ошибок при вводе данных. Разбив же данные по таблицам, можно в значительной степени избе-

жать этих трудностей.

Через связь, определенную между этими таблицами, можно узнать

- сколько альбомов выпустила группа;
- выпускались ли альбомы у фирмы ЕМІ;
- в каком году было выпущено максимальное количество альбомов и т.п.

Реляционные базы данных удобны еще и тем, что для получения ответов на различные запросы существует разработанный математический аппарат, который называется исчислением отношений или реляционной алгеброй. Ответы на запросы получаются путем «разрезания» и «склеивания» таблиц по строкам и столбцам. При этом ясно, что ответы также будут иметь форму таблиц.

Надо отметить, что база данных - это, собственно, хранилище информации и не более того. Однако, работа с базами данных трудоемкая и утомительная. Для создания, ведения и осуществления возможности коллективного пользования базами данных используются программные средства, называемые системами управления базами данных (СУБД).

6.4. СОСТАВ И ФУНКЦИИ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

База данных предполагает наличие комплекса программных средств, обслуживающих эту базу данных и позволяющих использовать содержащуюся в ней информацию. Такие комплексы программ называют СУБД. СУБД - это программная система, поддерживающая наполнение и манипулирование данными, представляющими интерес для пользователей при решении прикладных задач. Иными словами, СУБД является интерфейсом между базой данных и прикладными задачами.

Ниже перечислены основные функции СУБД.

1. **Определение данных** - определить, какая именно информация будет храниться в базе данных, задать свойства данных, их тип (например, число цифр или символов), а также указать, как эти данные связаны между собой. В некоторых случаях есть возможность задавать форматы и критерии проверки данных.

2. **Обработка данных** - данные могут обрабатываться самыми различными способами. Можно выбирать любые поля, фильтровать и сортировать данные. Можно объединять данные с другой, связанной с ними, информацией и вычислять итоговые значения.

3. **Управление данными** - можно указать, кому разрешено знакомиться с данными, редактировать их или добавлять новую информацию. Можно также определять правила коллективного доступа.

Входящие в состав современных СУБД средства совместно выполняют следующие функции:

- *описание* данных, их структуры (обычно описание данных и их структуры происходит при инициировании новой базы данных или добавлении к существующей базе новых разделов (отношений); описание данных необходимо для контроля корректности использования данных, для поддержания целостности базы данных);

- *первичный ввод, пополнение* информации в базе данных;
- *удаление* устаревшей информации из базы данных;
- *корректировку* данных для поддержания их актуальности;
- *упорядочение (сортировку)* данных по некоторым признакам;

- *поиск информации* по некоторым признакам (для описания запросов имеется специальный язык запросов, он обеспечивает также интерфейс между базой данных и прикладными программами пользователей, позволяет этим программам использовать базы данных);

- *подготовку и генерацию отчетов* (средства подготовки отчетов позволяют создавать и распечатывать сводки по заданным формам на основе информации базы данных);

- *защиту информации и разграничение доступа* пользователей к ней (некоторые разделы базы данных могут быть закрыты для пользователя совсем, открыты только для чтения или открыты для изменения; кроме того, при многопользовательском режиме работы с базой данных необходимо, чтобы изменения вносились корректно; для сохранения целостности данных служит механизм транзакций при манипулировании данными - выполнение манипуляций небольшими пакетами, результаты каждого из которых в случае возникновения некорректности операций «откаты-

ваются» и данные возвращаются к исходному состоянию);

- *резервное сохранение и восстановление* базы данных, которое позволяет восстановить утраченную при сбоях и авариях аппаратуры информацию базы данных, а также накопить статистику работы пользователей с базой данных;

- *поддержку интерфейса с пользователями*, который обеспечивается средствами ведения диалога (по мере развития и совершенствования СУБД этот интерфейс становится все более дружелюбным; дружелюбность существующих средств интерфейса предполагает

- наличие развитой системы помощи (подсказки), к которой в любой момент может обратиться пользователь, не прерывая сеанса работы с компьютером и базой данных;

- защиту от необдуманных действий, предупреждающую пользователя и предотвращающую потерю информации в случае поспешных или ошибочных команд;

- наличие нескольких вариантов выполнения одних и тех же действий, из которых пользователь может выбрать наиболее удобные для себя, соответствующие его подготовке, квалификации, привычкам;

- тщательно продуманную систему ведения человеко-машинного диалога, отображение информации на дисплее, использование клавиш клавиатуры). В настоящее время выделяют пять уровней проблематики систем управления базами данных:

- реляционные базы данных, 1970 - 90 гг.;

- объектно-ориентированные базы данных, 1980 - 90 гг.;

- интеллектуальные базы данных, 1985 - 90 гг.;

- распределенные базы данных, начало 1990 гг.;

- базы данных мультимедиа и виртуальной реальности настоящего времени.

Архитектурно СУБД состоит из двух основных компонентов; **языка описания данных** (ЯОД), позволяющего создать схему описания данных в базе, и **языка манипулирования данными** (ЯМД), выполняющего операции с базой данных (наполнение, обновление, удаление, выборку информации). Данные языки могут быть реализованы в виде тренажеров или интерпретаторов. Помимо ЯОД и ЯМД к СУБД следует отнести **средства** (или языки) **подготовки отчетов** (СПО), позволяющие подготовить сводки (отчеты) на основе информации, найденной в базе данных, по заданным формам.

Язык описания данных (ЯОД) - это язык высокого уровня *декларативного* (непроцедурного) типа, предназначенный для формализованного описания типов данных, их структур и взаимосвязей. Исходные тексты описания данных на этом языке после трансляции отображаются в управляющие таблицы, задающие размещение в памяти ЭВМ и связи между собой рассматриваемых данных. В соответствии с этими описаниями СУБД находит в базе требуемые данные, правильно преобразует их и передает, например, в прикладную программу пользователя, которой они потребовались. При записи данных в базу СУБД по этим описаниям определяет место в памяти ЭВМ, куда их требуется поместить, преобразует к заданному виду и устанавливает необходимые связи.

Язык манипулирования данными (или язык запросов) представляет собой систему команд, например, следующего типа:

- произвести выборку данного, значение которого удовлетворяет заданным условиям;

- произвести выборку всех данных определенного типа, значения которых удовлетворяют заданным условиям;

- найти в базе позицию данного и поместить туда новое значение (или удалить данное) и т.д.

Широкое распространение имеют СУБД для персональных компьютеров типа DBASE (DBASE III, IV, FoxPro, Paradox), Clipper, Clarion. Эти СУБД ориентированы на однопользовательский режим работы с базой данных и имеют очень ограниченные возможности. Языки подобных СУБД представляют собой сочетание команд выборки, организации диалога, генерации отчетов. В связи с развитием компьютерных сетей, в которых персональные компьютеры выступают в качестве развитых (интеллектуальных) терминалов, новые версии СУБД все в большей степени включают в себя возможности описанного ниже языка манипулирования данными SQL.

В последнее время стали среди СУБД популярными ACCESS (входит в состав MS Office), Lotus, Oracle.

Язык манипулирования данными SQL

Рассмотрим в качестве примера языка манипулирования данными некоторые команды языка SQL (от английских слов Structured Query Language), ставшего классическим языком реляционных баз данных.

Простейшая операция выборки представляется командой SELECT - FROM -WHERE (выбрать - из - где):

```
select <список атрибутов>
from <отношение>
where <условие>.
```

Например, если необходимо из отношения «Успеваемость», имеющего схему:

Успеваемость (ФИО_студента, Дисциплина, Оценка, Дата, Преподаватель)

произвести выборку данных о том, какие оценки студент Иванов И.И. получил и по каким предметам, надо задать команду:

```
select      Дисциплина, Оценка
from        Успеваемость
where       ФИО_студента = «Иванов И. И.».
```

Часть команды «where» не является обязательной. Например, можно получить список всех студентов из отношения «Успеваемость» с помощью следующей команды:

```
select      unique ФИО_студента
from        Успеваемость.
```

Ключевое слово unique позволяет исключить из результата дубликаты значений атрибута. Выбрать полностью информацию из таблицы можно с помощью команды

```
select      *
from        Успеваемость.
```

Условие, следующее за «where», может включать операторы сравнения =, <>, >=, <, <=, булевы операторы AND, OR, NOT, а также скобки для указания желаемого порядка операции. Например, выбрать из таблицы «Успеваемость» фамилии студентов, сдавших на "5" экзамен по информатике, можно с помощью команды

```
select      ФИО_студента
from        Успеваемость
where       Дисциплина = «Информатика» AND Оценка=5.
```

Выборка может быть и вложенной, когда необходимо использовать в условии результаты другой выборки. Например, если надо из отношения «Успеваемость» выбрать только студентов физико-математического факультета, пользуясь отношением «Студент», то команда select может выглядеть так:

```
select      ФИО_студента
from        Успеваемость
where       ФИО_студента is in
(select     Фамилия
from        Студент
where       Ф_т = «физмат»).
```


Здесь «is in» является представлением оператора принадлежности элемента множеству. Можно также использовать операторы «is not in» («не принадлежит множеству»), «contains» - содержит, «does not contains» - не содержит. Смысл выражения «A contains B» (A содержит B) тот же, что и выражения «B is in A» (B принадлежит множеству A). Помимо слов select, from, where в команде выборки можно использовать и другие служебные слова, например:

order by <атрибут> asc	- определяет сортировку результата выборки в порядке возрастания (asc) или убывания (desc) значения атрибута;
group by <атрибут1>	- группирует данные по значениям атрибута;
having set <атрибут2>	
minus	- операция вычитания множеств (данных выборок).

Помимо команды выборки select, язык SQL имеет команды, позволяющие обновлять данные (update), вставлять (insert) и удалять (delete). Например, если студенты переводятся со 2-го курса на третий, информацию можно обновить командой

```
update      Студент
set         Курс=3
where       Курс=2.
```

Если атрибут «Семенов С.С.» сдал экзамен по информатике на «5» 15 января 1996 г. преподавателю Петрову П.П., то информация об этом может быть добавлена в таблицу «Успеваемость» командой

```
insert into Успеваемость:
<«Семенов С.С.», «Информатика», 5,15/01/96, Петров П.П.>.
```

Оператор insert может быть использован для включения одной строки (как в этом примере) или произвольного числа строк, определенных списком кортежей, заключенных в скобки, или операций выборки select из какой-либо другой таблицы. Команда delete используется для удаления информации из таблицы. Например,

```
delete Успеваемость
where Оценка=2
```

позволяет удалить информацию о студентах, получивших 2 (в случае их отчисления).

Существенно расширяют возможности языка библиотечные функции, такие как count (подсчет), sum (суммирование), avg (среднее), max и min.

Например, подсчитать число студентов в таблице «Студент»: select count (*) from Студент.

6.5. ПРИМЕРЫ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

СУБД DBASE

СУБД типа DBASE позволяют работать с реляционными базами данных (БД), структура которых состоит из трех элементов:

- число полей БД;
- характеристика каждого поля;
- число записей в БД.

Каждое поле имеет следующие характеристики:

Field name	Type	Width	Dec

(имя поля) (тип) (ширина) (дес.знаки).

Field name - может состоять из набора символов, но без пробелов.

Type - в системах типа DBASE имеется 5 типов полей:

 C (Character) - символьный (текстовый) тип;

 N (Numerical) - числовой тип;

 L (Logical) - логический тип;

 D (Date) - поле дат, содержит даты в виде dd/mm/yy;

 M (Memo) - поле памяти, содержит большой текст (файл).

Width - обозначает допустимую ширину поля.

Dec - используется для числовых полей и определяет точность задаваемых чисел.

DBASE создает следующие типы файлов:

.dbf - файлы с записями БД;

.prg - файлы с текстами программ;

.frm - файлы структуры форматных отчетов;

.ndx - индексные файлы, сортирующие записи по определенному ключу;

.mem - файлы данных переменной Mem.

Запуск СУБД осуществляется **из** операционной системы exe-файлом (db.exe, foxdb и т.п.), выход-командой **.Quit.**

Теперь опишем кратко основные команды СУБД.

Создание БД осуществляется командой **Create.**

Create (например, «Абитуриент»)

После ввода этой команды на экране появится форма:

Field name	Type	Width	Dec
(имя поля)	(тип)	(ширина)	(дес.знаки).

.001

В соответствии с этой формой создадим структуру таблицы:

001 ФИО, C, 18

002 год_рожд, C, 7

003 район. C, 13

004 адрес. C, 100

005 группа, C, 3

006 оценка 1,N,3

007 оценка2,N, 3

008 оценка3,N, 3

Теперь можно начать заполнение таблицы записями.

В случае заполнения записями уже существующей базы данных, необходимо предварительно эту базу командой Use сделать активной:

.Use Абитуриент (use - использовать),

.Append (добавить)

Данные вводят в карточки, имеющие следующую форму:

Запись #00001

ФИО:

год_рожд:

район :

адрес:

группа:
оценка1:
оценка2:
оценка3:

Например,

Запись	#00005
ФИО:	Семенов Сергей Викторович
год_рожд:	1980
район:	Туруханский
адрес:	ул. Декабристов, д. 12, кв.23
группа:	И2
оценка1:	5
оценка2:	4
оценка3:	4

Запись можно ввести в определенное место БД, введя одну из команд:

.Insert (вставить)

или

.Insert before .

Перемещение по таблице и просмотр записей БД осуществляется командами:

Go top	- (идти наверх) установка указателя на первую запись,
Go bottom	- (идти вниз) установка указателя на последнюю запись;
List	- (список) просмотр всех записей БД;
Display	- (отобразить) просмотр записи, на которой находится указатель,
Browse	- (просмотреть) помимо просмотра позволяет редактировать записи БД.

Редактирование записей позволяют проводить следующие команды:

Edit N	- редактирование записи с номером N;
Change	- (поменять) изменения только в некоторых полях или записях, удовлетворяющих заданным условиям;
Delete	- (удалить) стирание ненужных записей;
Copy	-(копировать) копирование записей.

Изменить структуру БД можно командой **Modify**. Ниже предложен перечень команд, осуществляющих обработку данных:

Report form	- (отчет, форма) создание отчетов;
Sort	- (сортировка) упорядочение БД по какому-либо ключу;
Index	- (индекс) индексирование БД;
Find	- (найти) поиск в БД.

Работу с несколькими БД помогают вести команды:

Select	- (выбор) сделать активной какую-либо БД;
Update	- (расширить) передача данных из одной БД в другую;
Join to	- (присоединить) соединение целых БД.

Для осуществления интерактивности БД используют команды ввода и вывода:

Wait	- (ожидание) пауза, приостановка;
Input	- (вход) ввод данных;
Say	- (сказать) вывод информации;
Read	- (читать) ввод данных.

СУБД Microsoft Access

Access - в переводе с английского означает «доступ». MS Access - это функционально полная реляционная СУБД. Кроме того, MS Access одна из самых мощных, гибких и простых в использовании СУБД. В ней можно создавать большинство приложений, не написав ни единой строки программы, но если нужно создать нечто очень сложное, то на этот случай MS Access предоставляет мощный язык программирования - Visual Basic Application.

Популярность СУБД Microsoft Access обусловлена следующими причинами:

- Access является одной из самых легкодоступных и понятных систем как для профессионалов, так и для начинающих пользователей, позволяющая быстро освоить основные принципы работы с базами данных;

- система имеет полностью русифицированную версию;

- полная интегрированность с пакетами Microsoft Office: Word, Excel, Power Point, Mail;

- идеология Windows позволяет представлять информацию красочно и наглядно;

- возможность использования OLE технологии, что позволяет установить связь с объектами другого приложения или внедрить какие-либо объекты в базу данных Access;

- технология WYSIWIG позволяет пользователю постоянно видеть все результаты своих действий;

- широко и наглядно представлена справочная система;

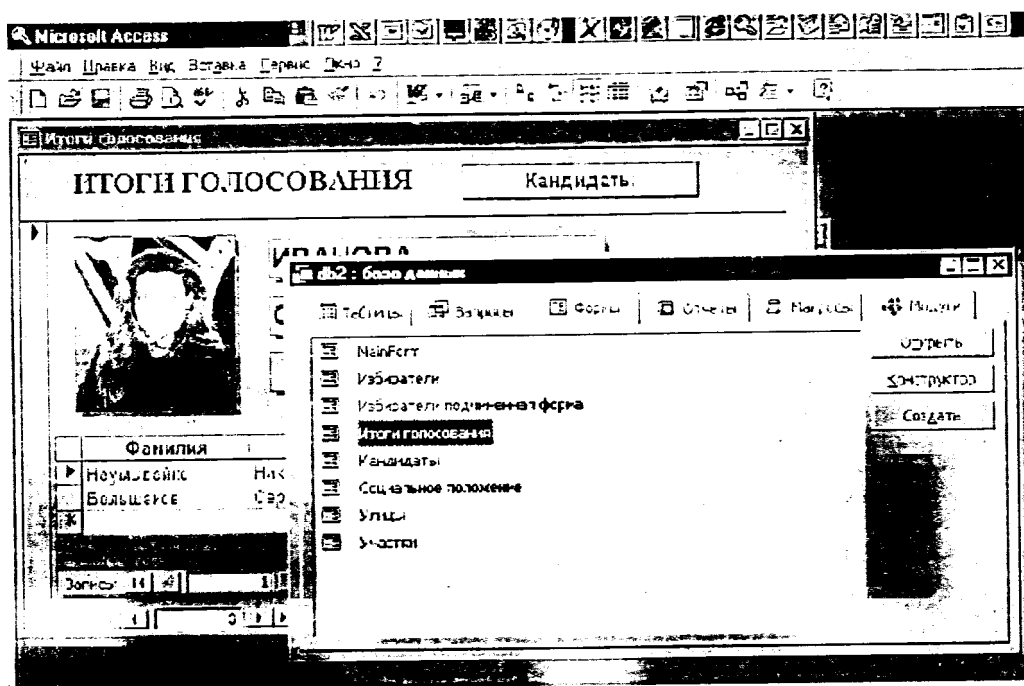
- существует набор «мастеров» по разработке объектов, облегчающий создание таблиц, форм и отчетов.

Запустить систему Access можно несколькими способами:

- запуск с помощью главного меню в WINDOWS 95;

- запуск с помощью ярлыка на панели инструментов.

После запуска системы появится главное окно Access, рис. 2.24. Здесь можно открывать другие окна, каждое из которых по-своему представляет обрабатываемые данные. Ниже приведены основные элементы главного окна Access, о которых необходимо иметь представление.



В строке заголовка отображается имя активной в данный момент программы. Строка заголовка главного окна Access всегда отображает имя программы MICROSOFT Access.

Пиктограмма системного меню - условная кнопка в верхнем левом углу главного окна практически любого приложения. После щелчка на этой пиктограмме появляется меню, которое позволяет перемещать, разворачивать, сворачивать или закрывать окно текущего приложения и изменять его размеры. При двойном щелчке на пиктограмме системного меню работа приложения завершается.

Полоса меню содержит названия нескольких подменю. Когда активизируется любое из этих названий, на экране появляется соответствующее подменю. Перечень подменю на полосе Access и их содержание изменяются в зависимости от режима работы системы.

Панель инструментов - это группа пиктограмм, расположенных непосредственно под полосой меню. Главное ее назначение - ускоренный вызов команд меню. Кнопки панели инструментов тоже могут изменяться в зависимости от выполняемых операций. Можно изменять размер панели инструментов и передвигать ее по экрану. Также можно отобразить, спрятать, создать новую панель инструментов или настроить любую панель инструментов.

В левой части **строки состояния** отображается информация о том, что вы делаете в настоящее время.

Окно базы данных появляется при открытой базе данных. В нем сосредоточены все «рычаги управления» базой данных. Окно базы данных используется для открытия объектов, содержащихся в базе данных, таких как таблицы, запросы, отчеты, формы, макросы и модули. Кроме того, в строке заголовка окна базы данных всегда отображается имя открытой базы данных.

С помощью **вкладки объектов** можно выбрать тип нужного объекта (таблицу, запрос, отчет, форму, макрос, модуль). Необходимо сказать, что при открытии окна базы данных всегда активизируется вкладка-таблица и выводится список доступных таблиц базы данных. Для выбора вкладки других объектов базы данных нужно щелкнуть по ней мышью.

Условные кнопки, расположенные вдоль правого края окна базы данных, используются для работы с текущим объектом базы данных. Они позволяют создавать, открывать или изменять объекты базы данных.

К основным объектам Access относятся таблицы, запросы, формы, отчеты, макросы и модули.

Таблица - это объект, который определяется и используется для хранения данных. Каждая таблица включает информацию об объекте определенного типа. Как уже известно, таблица содержит поля (столбцы) и записи (строки). Работать с таблицей можно в двух основных режимах: в режиме конструктора и в режиме таблицы.

В режиме конструктора задается структура таблицы, т.е. определяются типы, свойства полей, их число и названия (заголовки столбцов). Он используется, если нужно изменить структуру таблицы, а не хранящиеся в ней данные. В этом режиме каждая строка верхней панели окна соответствует одному из полей определяемой таблицы.

Режим таблицы используется для просмотра, добавления, изменения, простейшей сортировки или удаления данных. Чтобы перейти в режим таблицы, надо дважды щелкнуть мышью по имени нужной таблицы в окне базы данных (или, выделив в окне БД имя нужной таблицы, воспользоваться кнопкой открытого окна БД).

Из режима конструктора перейти в режим таблицы можно, щелкнув по кнопке таблицы на панели инструментов.

В режиме конструктора и в режиме таблицы перемещение между полями осуществляется с помощью клавиши TAB, а также вверх или вниз по записям с помощью клавиш, но в большинстве случаев пользоваться мышью гораздо удобнее.

Вследствие того, что в таблицах, как правило, содержится большое количество записей, размещение всех их на экране невозможно. Поэтому для перемещения по таблице используют полосы прокрутки, расположенные в нижней и правой части окна. Левее нижней полосы прокрутки выводится номер текущей записи и общее число записей таблицы. Для перехода к записям с нужным номером необходимо активизировать поле *Номера записи*, щелкнув по нему, или нажать клавишу F5, после чего набрать на клавиатуре новый номер записи и затем нажать клавишу <Enter>.

Запрос - это объект, который позволяет пользователю получить нужные данные из одной или нескольких таблиц. Можно создать запросы на выбор, обновление, удаление или на добавление данных. С помощью запросов можно создавать новые таблицы, используя данные уже существующих одной или нескольких таблиц.

По сути дела, запрос - это вопрос, который пользователь задает Access о хранящейся в базе данных информации. Работать с запросами можно в двух основных режимах: в режиме конструктора и в режиме таблицы.

Здесь надо вспомнить о том, что ответы на запросы получаются путем «разрезания» и «склеивания» таблиц по строкам и столбцам, и что ответы будут также иметь форму таблиц. В режиме конструктора формируется вопрос к базе данных.

Форма - это объект, в основном, предназначенный для удобного ввода отображения данных. Надо отметить, что в отличие от таблиц, в формах не содержится информации баз данных (как это может показаться на первый взгляд). Форма - это всего лишь формат (бланк) показа данных на экране компьютера. Формы могут строиться только на основе таблиц или запросов. Построение форм на основе запросов позволяет представлять в них информацию из нескольких таблиц.

В форму могут быть внедрены рисунки, диаграммы, аудио (звук) и видео (изображение).

Режимы работы с формой:

- *режим формы* используется для просмотра и редактирования данных; предоставляет дружелюбную среду для работы с данными и удобный дизайн их представления на экране;

- *режим конструктора форм* необходим, если необходимо изменить определение формы (структуру или шаблон формы, а не представленные в ней данные), надо открыть форму в режиме конструктора;

- *режим таблицы* позволяет увидеть таблицу, включающую все поля формы; чтобы переключиться в этот режим при работе с формой, надо нажать кнопку таблицы на панели инструментов.

Отчет - это объект, предназначенный для создания документа, который впоследствии может быть распечатан или включен в документ другого приложения. Отчеты, как и формы, могут создаваться на основе запросов и таблиц, но не позволяют вводить данные.

Режимы работы с отчетом:

Режим предварительного просмотра позволяет увидеть отчет таким, каким он будет воплощен при печати. Для того чтобы открыть отчет в режиме предварительного просмотра, надо

- щелкнуть по вкладке *Отчеты*,
- кнопкой выбрать необходимый отчет в окне базы данных;
- щелкнуть по кнопке *Просмотра*.

Режим конструктора предназначен для изменения шаблона (структуры отчета).

Макрос - это объект, представляющий собой структурированное описание одного или нескольких действий, которые должен выполнить Access в ответ на определенное событие. Например, можно определить макрос, который в ответ на выбор некоторого элемента в основной форме открывает другую форму. С помощью другого макроса можно осуществлять проверку значения некоторого поля при изменении его содержания. В макрос можно включить дополнительные условия для выполнения или невыполнения тех или иных включенных в него действий. Возможно также из одного макроса запустить другой макрос или функцию модуля.

Работа с формами и отчетами существенно облегчается за счет использования *макрокоманд*. В MS Access имеется свыше 40 макрокоманд, которые можно включать в макросы. Макрокоманды выполняют такие действия, как открытие таблиц и форм, выполнение запросов, запуск других макросов, выбор опций из меню, изменение размеров открытых окон и т.п. Макрокоманды позволяют нажатием одной (или нескольких одновременно) кнопки выполнять комплекс действий, который часто приходится выполнять в течение работы. С их помощью можно даже осуществлять запуск приложений, поддерживающих динамический обмен данными (DDE), например MS Excel, и производить обмен данными между вашей базой данных и этими приложениями. Один макрос может содержать несколько макрокоманд. Можно также задать условия выполнения отдельных макрокоманд или их набора.

Модуль - объект, содержащий программы на MS Access Basic, которые позволяют разбить процесс на более мелкие действия и обнаружить те ошибки, которые невозможно было бы найти с

использованием макросов.

Завершив работу с Access (или с ее приложением), надо корректно закончить сеанс. Простое выключение компьютера - плохой метод, который может привести к возникновению проблем. При работе WINDOWS приложения используют множество файлов, о существовании которых пользователь может даже не подозревать. После выключения машины эти файлы останутся открытыми, что в будущем может сказаться на надежности файловой системы жесткого диска.

Безопасно выйти из Access можно несколькими способами:

- двойным щелчком мыши на пиктограмме системного меню в строке заголовка главного окна Access;
- из меню Access выбором пункта *Файл Выход*,
- нажатием комбинации клавиш Alt + F4.

Контрольные вопросы и задания

1. Каково назначение программ, входящих в состав СУБД?
2. Какой интерфейс можно считать дружественным?
3. Какие компоненты можно выделить в составе СУБД?
4. В чем состоят функции языков описания и манипулирования данными?
5. Охарактеризуйте основные команды языка SQL.
6. Как с помощью команд SQL задать поиск в базе данных?
7. Как с помощью команд SQL модифицировать базу данных?
8. Используя СУБД типа DBase разработайте базы данных:
 - а) телефонный справочник;
 - б) каталог программного обеспечения персонального компьютера;
 - в) домашняя библиотека.
9. Используя СУБД Access разработайте БД «Музыкальная энциклопедия».

§ 7. ЭЛЕКТРОННЫЕ ТАБЛИЦЫ

7.1. НАЗНАЧЕНИЕ И ОСНОВНЫЕ ФУНКЦИИ ТАБЛИЧНЫХ ПРОЦЕССОРОВ

Как показала практика, решение многих задач экономического характера на языках высокого уровня с использованием всего арсенала приемов и методов профессионального программирования - сложное и громоздкое дело. Понадобился принципиально иной подход, и он был найден и воплощен в виде электронных таблиц - инструмента, доступного непрофессионалам. Основная область применения электронных таблиц - это те сферы человеческой деятельности, где информация предоставляется в виде прямоугольных таблиц (планово-финансовых и бухгалтерских документов, учета материальных ценностей и др.), требующих при обработке проведения математических расчетов, откуда, по-видимому, и возник термин «табличный процессор». Отметим, что реляционные базы данных, также представляемые с помощью таблиц, к расчетам, как правило, не приспособлены.

В настоящее время известно много вариантов электронных таблиц: АБАК, Варитаб-86. Суперплан, Multiplan, SuperCalk, QuattroPro, Excel, Lotus 1-2-3 и др. Принципиально все они представляют табличный процессор и разнятся лишь интерфейсом и сервисными возможностями.

Электронная таблица (ЭТ) - это прямоугольная матрица, состоящая из ячеек, каждая из которых имеет свой номер, рис. 2.25.

A	B	C	D	E	F
1					
2		C2			
3	B3				
4					

Рис. 2.25. Электронная таблица

Номер ячейки определяется обычным координатным способом, например, ячейка B3 и т.д.

Группа ячеек (диапазон) задается через двоеточие, например, B3:D4 (или B3..D4) и образует прямоугольник, включающий ячейки B3, C3, D3, B4, C4, D4.

В каждую из ячеек можно занести *число*, *формулу* (арифметическое выражение) или *текст*. Если в ячейку ЭТ записана формула, то в исходном состоянии на экране отображается значение этой формулы, а не она сама. Операндами формулы могут быть математические функции, константы, номера ячеек (содержимое ячейки с указанным номером). Ячейка ЭТ имеет сложную «многослойную» структуру, в ней может стоять ссылка на другую ячейку, значение которой является результатом вычислений по другой формуле и т.д.

Примеры функций:

sum(A2:A8) - сумма значений всех ячеек от A2 до A8;

sin(D5) - синус числа из ячейки D5;

cos(F3) - косинус числа из ячейки F3.

Пример формулы:

2.7. * A6 + cos (sum (D5:F7))

Приведенная формула означает, что мы хотим получить результат следующих вычислений: произведение числа из ячейки A6 на 2.7 сложить с косинусом угла, который является суммой чисел из ячеек D5, E5, F5, D6, E6, F6, D7, E7, F7.

Данные, входящие в таблицы, можно автоматически представлять в виде графиков, диаграмм, гистограмм и т.д.

Пользователь работает в диалоге со специальной программой, которая позволяет заполнять ячейки нужным ему содержимым (текстами, числами или формулами для расчетов); очищать их, копировать и удалять, сортировать (т.е. располагать клетки, а также строки и столбцы из них, в определенном порядке); производить вычисления над всей таблицей или ее частью, сохранять таблицу на диске и распечатывать частично или полностью на бумагу и т.д.

Приведем пример, иллюстрирующий возможности ЭТ.

Формирование зарплатной ведомости.

Так выглядят исходные данные для заполнения электронной таблицы:

	A	B	C	D	E
1	Зарплатная ведомость фирмы «Рога и копыта»				
2	ФИО	Оклад	Начисление	Налог	Всего
3	Балаганов А	1500	B3* .6	C3*0.12	C3-D3
4	Бендер О	3000	B4* .6	C4*0.12	C5-D5
5	Паниковский М.	1000	B5* .6	C5*0.12	C5-D5

Здесь мы имеем дело с тремя типами содержимого ячеек: текст, число, формула. Ввод исходных данных происходит в командной строке. После заполнения таблицы мы увидим на экране:

	A	B	C	D	E
1	Зарплатная ведомость фирмы «Рога и копыта»				
2	ФИО	Оклад	Начисление	Налог	Всего
3	Балаганов А.	1500	2400	288	2112
4	Бендер О.	3000	4800	576	4224
5	Паниковский М.	1000	1600	192	1408

Обычно работник бухгалтерии, поправив одну из цифр, вынужден был исправлять весь комплект взаимосвязанных документов, куда явно или неявно входил исправленный параметр. С помощью ЭТ такое изменение может быть учтено мгновенно и всюду.

7.2. ЭЛЕКТРОННЫЕ ТАБЛИЦЫ SUPERCALC

Общие сведения

Одним из популярных табличных процессоров под DOS для компьютеров PC-286, -386 является SuperCalc-4 (SC-4).

SC4 позволяет работать с семью типами диаграмм и графиков, позволяет вводить различные обозначения, шкалы переменных, заголовки. В SC-4 можно создавать базы данных (БД), имеются простейшие средства, характерные для систем управления БД. Кроме того, SC-4 располагает средствами для перевода информации к виду, доступному из текстовых редакторов, систем управления БД (например, семейства DBASE) и других программных средств.

После загрузки системы на экране появляется рекламная заставка фирмы-разработчика и далее после нажатия любой клавиши - пустая электронная таблица. ЭТ, созданные пользователем, записываются на диск в специальном оригинальном формате и имеют по умолчанию стандартное расширение .cal. Эти файлы в неизменном виде могут обрабатываться только системами SuperCalc-4 и SuperCalc-5.

Электронная таблица SC-4 состоит из клеток, образующих строки (rows) и столбцы (columns). Столбцы обозначены одно- и двухсимвольными буквами латинского алфавита: A, B, C, ..., Z, AB, ..., AZ, BA, ..., BZ, ..., IA, ..., IU. Максимальное число столбцов 255. Строки обозначены номерами от 1 до 9999. В обозначении каждой клетки указывают координаты столбца и строки. Например: A1, B20, IA1. Такое обозначение клетки еще называют адресом клетки. В каждый момент времени одна из клеток является активной (АК). Она высвечивается на экране при помощи указателя, которым можно управлять. Активная клетка доступна пользователю для чтения и записи данных. Для быстрого перемещения указателя к краю ЭТ используют одновременное нажатие клавиши END и стрелок (к верхнему краю, к нижнему, к левому и к правому). Для листания ЭТ по страницам используют клавиши PageUp (страница вверх), PageDown (страница вниз), Ctrl + <=, Ctrl + ==> (страницы влево или вправо).

ЭТ имеет обрамление (верхняя строка и левый столбец) с именами строк и столбцов. Сами клетки составляют рабочую область ЭТ. Напомним, что на экране видна лишь часть таблицы. В нижней части экрана расположены четыре служебные строки:

- в первой строке отображается адрес и содержимое АК, а также направление движения указателя АК;
- во второй строке содержится информация об ЭТ;
- в третьей строке вводятся данные или команды;
- в четвертой строке содержатся подсказки и дополнительная информация о режиме работы ЭТ, назначении функциональных клавиш или пунктов меню команд.

Объекты, с которыми работает SC4: клетки, столбец, строка, диапазон столбцов (например A:C), диапазон строк (например 4:7) и блок клеток. Блок клеток задается адресами левой верхней и правой нижней клеток. В качестве разделителя используется двоеточие или точка, например, A3:B5 или A3.B5.

Список - один или более адресов объектов, разделенных запятыми.

Ссылки - адреса клеток, используемые в качестве имен переменных в формулах.

Обычное обозначение адресов является относительным, так как оно показывает расстояние от клетки, в которой содержится формула, до клетки, на которую в этой формуле есть ссылка. Например, пусть в клетке A3 хранится формула A2 + 1 (т.е. клетка A3 должна принять значение клетки A2, увеличенное на единицу). При копировании этой формулы в другие клетки, в них будет возникать не ссылка A2, а ссылка на клетку, расположенную выше, подобно тому как A2 есть клетка, расположенная над клеткой A3 - так называемое, «копирование с настройкой», значительно ускоряющее формирование ЭТ при решении многих задач.

Для того, чтобы ссылка не изменялась, используют абсолютный адрес. В этом адресе перед номером столбца и строки должен стоять знак \$. Например, \$A\$2 - не перенастраивается ни номер столбца, ни номер строки; \$A2 - не перенастраивается только номер столбца, номер строки остается относительным; A\$2 - здесь относителен номер столбца.

Содержимым клетки может быть текст, повторяющийся текст и формула:

- формула используется для вычислений, строится из чисел, математических операторов и

функций; длина формулы до 241 символа (частный вид формулы -число);

- текст - может содержать любой символ клавиатуры, его длина должна быть не более 241 символа, для введения текста набор следует начинать с символа S или кавычки ("");

- повторяющийся текст - начинается с апострофа (') и распространяется на все свободные клетки строки (обычно используется для прочерчивания линий).

Важной особенностью ЭТ является тот факт, что как только в клетку внесено новое содержимое, автоматически происходит изменение значений всех клеток, содержащих на нее ссылку.

SC-4 поддерживает два вида математических операторов: арифметические и операторы отношения. Арифметические операторы +, -, *, / имеют стандартное назначение, Операторы % (расчет процента), ** или ^ - возведение в степень. Операторы отношения: <, >, =, <=, >= (меньше, больше, равно, меньше или равно, больше или равно).

Имеются следующие виды функций: арифметические и тригонометрические, логические, календарные, статистические, специальные, финансовые, индексные.

После ввода символа "/" в первой и второй служебных строках появляется меню команд. Для выбора команды надо переместиться на ее имя и нажать клавишу ввода или ввести только первую букву ее имени. В ответ программа выводит полное имя команды:

/Blank	-очистка клетки, группы клеток, всей таблицы или описания графиков;
/Insert	- вставка пустых строк/колонок;
/View	- визуализация данных в графическом режиме;
/Move	- перенос-вставка существующих строк/колонок с указанной позиции;
/Global	- задание общих режимов или режимов пересчета таблицы;
/eXecute	- исполнение командного файла (xqt. файл);
/Copy	- копирование содержания клеток или описания графиков;
/Zap	-удаление таблицы и значения форматных характеристик из памяти;
/Load	- загрузка таблицы или ее части с диска в рабочую область памяти;
/Window	- установка режима «два окна»;
/Output	- вывод отображения данных или содержимого клеток на экран, на диск или на печать;
/Edit	- редактирование содержания клетки;
/Arrange	- сортировка данных (строк, колонок);
/Title	- фиксация заголовка и/или левых колонок таблицы,
/Delete	- удаление колонки (строки), файла;
/Format	- установление форматных характеристик отображения дан-ных на уровне

клетки, строки, колонки или всей таблицы;

/Save	- сохранение текущего содержания таблицы на диске;
/Quit	- завершение сеанса работы с программой;
/Unprotect	- снятие защиты клеток;
/Protect	- установка защиты клеток;
/Name	- задание имени для диапазона клеток;
//Data	- предлагает дополнительные команды для работы с базой данных;
//Export	- пересылка файлов из Суперкалка;
//Import	- пересылка файлов в Суперкалк;
//Macro	-создание макроопределений. Большинство команд имеют несколько уровней

возможных ответов. После ввода буквы команды, вместо списка команд, появляется подсказка с вариантами ответов, допустимыми для этой команды.

Пример: создание ЭТ «Штатное расписание»

Общий вид создаваемой таблицы:

	A	B	C	D	E	F	G	H	I	J	K
1				Школа №5							
2											
3	ФИО	Стаж	Разряд	Разрядный коэффици- ент	Оклад. Руб.	Число часов	Всего начис- лено	Подо- ходный налог	Проф- союзные сборы	Всего удер- жано	К вы- даче

4										
5										
6	Артемьева Т.Н.	5	10	3,30	86	20	124,22	14,91	0,12	15,03 109,19
7	Бердышева А.	7	10	3,30	86	30	186,33	22,36	0,19	22,55 163,78
8	Пришвина О.Н.	9	11	3,40	88	21	133,47	16,02	0,13	16,15 117,32
9	Веселова В.А.	16	12	3,45	89	24	154,27	18,51	0,15	18,67 135,60
10	Николаева С.Ф.	15	11	3,40	88	28	177,96	21,35	0,18	21,53 156,43
11	Левина Е.А.	10	12	3,45	89	18	115,70	13,88	0,12	14,00 101,70

Опишем порядок выполнения работы.

А. Оформление шапки таблицы.

1. Перемещаем курсор АК в клетку D1 и набираем с клавиатуры текст «Школа №5», нажимаем клавишу ввода.

2. В клетку A2 вводим повторяющийся текст: ' _ . Он должен начинаться с апострофа, за ним указывается тот символ, который должен повторяться. В нашем случае это знак подчеркивания. Во второй строке проведена линия до правого края таблицы. Очистите клетки таблицы от линии, начиная со столбца L. Для этого введите в клетку L2 повторяющийся текст, содержащий пробел. Вернитесь к левому краю таблицы (Ctrl <==).

3. В строки 3 и 4 введите заголовки столбцов, как показано в общем виде таблицы.

4. В пятой строке проведите линию, действуя аналогично п.2.

Б. Внесение данных.

1. Заполните столбец A. Так как по умолчанию ширина столбцов 9 символов, необходимо увеличить ширину столбца A. Выполните команду:

/F,CA,W,15.

Эта форма записи команды означает: нажатием клавиши «/» мы выходим в меню команд; клавишей F выбирается команда Format, нажатием C указывается область действия команды: Column (столбец); символ A автоматически появляется в командной строке, поскольку курсор АК находится в столбце A (или переведите его в этот столбец); далее выбирается опция Width (ширина) и указывается ширина столбца 15.

2. Внесите данные в столбцы B,C,D,F.

3. В клетку E6 внесем формулу для расчета оклада: (D6+1)*20. Обратите внимание, что в этой клетке сразу появилось числовое значение этой формулы. Если же этого не произошло, значит при наборе была ошибка и формула воспринята как текст. Для исправления ошибки нажать F2, удалить признак текста - кавычки - и внести исправления в формулу.

4. Аналогично п.3 занесите формулу для расчета суммы начисления:

E6*F6/18* 1.3 (эту формулу при желании можно сделать более точной).

5. В клетки H6 и I6 введите формулы 12% G6 и 1 % G6, соответственно.

6. В клетку J6 внесите формулу H6 + I6.

7. В клетку K6 внесите формулу G6 - J6.

8. Скопируйте формулу для расчета оклада из клетки E6 в блок клеток командой

E7:E11 :/C,E6,E7:E11.

Далее процесс аналогичен.

Во всех клетках автоматически появились числовые значения формул. В первой служебной строке можно увидеть их вид. Заметьте, что при копировании произошла автоматическая настройка адреса D6 на D7, D8, D9 и т.д.

/C,G6:K6,G7:K11.

Рассмотрим на примере этой ЭТ применение команд Title, Global, Window, Arrange, а также запись и чтение с диска.

1. Вся таблица не вмещается на экран. Можно зафиксировать столбец А, тогда при движении к последним столбцам ЭТ, например к столбцу с суммой начисления, столбец с фамилиями будет служить границей и не исчезнет с экрана. Для этого сначала курсор АК поместите на столбец А, а затем выполните команду

/Title, Vertical.

Передвиньте курсор АК в столбец Н и убедитесь в правильности выполненных действий. Опция Clear снимает все титульные значки. Выполните команду /Title, Clear.

Для фиксации шапки таблицы, надо поместить курсор АК на 5-ю строку и выполнить команду

/Title. Horizontal.

Если поместить курсор АК в клетку А5 и выполнить команду

/Title,Both,

то фиксируется и шапка таблицы и столбец А.

2. Поместите курсор АК в клетку А12. Выполните команду Window, Horizontal.

Экран разбился на два окна. Во втором окне при помощи, стрелок отобразите те же строки, что и в первом - в окнах можно высвечивать разные части одной и той же таблицы. Курсор АК сейчас находится во втором окне. Переход между окнами - клавиша F6. Настройте второе окно на режим отображения формул:

/Global,Formula.

Обратите внимание, что некоторые опции команды высвечиваются желтым цветом - отключены соответствующие им режимы работы ЭТ. Выбор этих опций означает переключения с пассивного режима работы на активный и наоборот. В нашем случае включается режим отображения формул. По умолчанию установлено несинхронное перемещение информации на экране, т.е. информация, смещаемая в одном окне, остается неподвижной во втором. После выполнения команды

Window, Synchronize

установится синхронный режим смещения (Un synchronize - несинхронный).

3. Запишите ЭТ на диск при помощи команды Save:/S . Во второй строке появляется запрос: Enter File Name (введите имя файла). Укажите имя файла, например PR1. Из опций A|| (вся таблица). Values (без формул), Part (часть таблицы) выберите A||. Назначение опций можно посмотреть при помощи клавиши-подсказки F1 в момент их высвечивания на экране. Команда записи на диск

/Save.PR1All.

4. Команда /Zap удаляет всю таблицу из памяти. Загрузить таблицу с диска можно командой Load:

/Load.PR1, AH.

5. Сортировка данных в таблице производится командой Arrange. Опция Row означает, что по значениям указанной строки будут сортироваться столбцы, опция Column - сортировка производится между строками по значениям столбца.

Отсортируем строки таблицы по столбцу А, в котором находятся фамилии сотрудников, в

соответствии с алфавитом. Выполните команду

/Arrange, Column, A,

но не нажимайте клавишу ввода. Обратите внимание на 2-ю служебную строку: "Enter Column; then <RETURN>, or <,> for Options" (введите колонку, затем ввод или запятая для опций). Вы должны нажать запятую. Это связано с тем, что не все строки нашей ЭТ должны сортироваться (в строках с 1-й по 5-ю находится шапка таблицы). На запрос "Enter Range" (введите область) укажите A6:K11 - можно воспользоваться удобствами режима Point. Далее из опций Ascending, Descending (по возрастанию, по убыванию) выберите Ascending. Из следующих опций Adjust, No-Adjust (с настройкой, без настройки формул) выберите опцию Adjust. Далее: Go. Общий вид команды

/Arrange, Column, A, A6:K 11, Ascending, Adjust, Go.

6. Отсортируем теперь таблицу по двум признакам: по убыванию разрядов, а внутри по возрастанию количества часов. Выполните команду:

/Arrange, Column, C, A6: K11, Descending, Adjust, Options, F, Ascending.

Средства макропрограммирования

Интегрированная система обработки электронных таблиц SC4 предоставляет пользователю средства макропрограммирования.

Макропрограммы хранятся, как правило, вместе с электронными таблицами и используются для автоматизации их обработки. Макропрограмма состоит из макросов. Макрос - самостоятельная структурная единица макропрограммы. Он обычно имеет имя и отделяется от остальных макросов пустыми клетками.

Макрос состоит из макрокоманд. С помощью макрокоманд можно записать любые действия пользователя, выполняемые в ручном режиме.

Работа с макросами включает в себя

- создание;
- поименование;
- запись на диск;
- отладку и исполнение.

Создавать макросы лучше всего непосредственно в ЭТ в обычном режиме ввода данных ENTRY.

Рекомендуется столбец A отводить для записи имен макросов, столбец B - для записи макрокоманд, а столбец C - для комментария, поясняющего действия макрокоманд.

Макрокоманды записывают в клетки как текст. Прописные и заглавные буквы не различают. Например,

/Ba1:a5~и/BA1:A5~

одна и та же команда. (Также как \a и \A - одно и то же имя или метка макроса). Значок ~ означает нажатие клавиши ввода. При вводе слэш-команд начинаем с символа ", причем вводим не полное название пункта меню, а выделенную букву.

Например, команда ввода

/Blank,al:a2

запишется в виде макрокоманды так:

"/Ba1:a2~.

Пример. Напишем макропрограмму, состоящую из трех макросов, которая будет выполнять

следующие действия по обработке таблицы:

- очищать клетки a1:c1 таблицы;
- запрашивать значение переменной x в клетку a1;
- если значение $x > 0$, то в клетке b1 выводить значение выражения $x * 0.25$, а иначе в клетке c1 значение выражения $x * .\backslash * 0.25$.

Наша макропрограмма в ЭТ будет выглядеть так:

	A	B	C
1			
2	\a	/Bal:cl	-Очистка клеток a1:c1
3		{Getnumber "x=",al}	Ввод значения x в a1
4		{if al>0} {Branch \b}	По условию, переход в \b
5		(Branch \c}	или в\c
6			
7	\b	{let blal *0.25}	b1=a1*0.25
8			
9	\c	{letclal*a1*0.25}	c1=a*a1*0.25
10			

В нашем примере три макроса \a, \b, \c отделяются друг от друга пустыми строками. Макрос с именем \a располагается в клетках b2:b5, макрос с именем \b в клетке b7, макрос с именем \c — в клетке b9. Выше описано создание макросов в режиме ENTRY. Кроме этого, макросы можно создавать в режимах LEARN и DIRECT. В режиме LEARN ваши действия автоматически записываются в виде макроса в LEARN — область, задаваемую командой

//Macro ,Learn, столбец.

Вход в этот режим осуществляется по нажатию клавиш Alt+F4. (Выход - повторное нажатие тех же клавиш.) Макрос, созданный в этом режиме, имеет большие размеры и сложен для восприятия и редактирования. Режим DIRECT (вход -Alt+F6, выход - повторное нажатие) является промежуточным между режимами ENTRY и LEARN.

Имена макросов лучше всего начинать с символа \ и далее одна из букв латинского алфавита. Макрос тогда очень просто запускается на выполнение: одновременное нажатие клавиш Alt+A запускает на выполнение макрос \a, Alt+B - макрос \b и т.д.

Чтобы текст в клетках столбца A воспринимался как имена (или метки) макросов, необходимо дать команду

- /Name, Labels, Right, A.

Эта команда назначает имена макросам, расположенным справа от столбца A. Макросы лучше всего записывать вместе с ЭТ по команде

/Save,имя ЭТ,A11.

Кроме этого, существует возможность записывать файлы с макросами в формате ASCII. В этом случае используется команда

//Macro,Write.

Эти файлы имеют стандартное расширение .xqt и могут создаваться в текстовых редакторах. Метки, макросы и комментарии записываются, тогда в один столбец. Первая строка файла должна содержать имя {Macro}. Эти файлы могут быть запущены на исполнение из SC по команде

//Macro.eXecute.имя.

Читаются такие файлы в SC по команде

//Macro,Read.

Графическое представление данных

Электронные таблицы имеют развитые возможности представления данных в графическом виде.

Рассмотрим принципы построения диаграмм на примере ЭТ «Показатели соревнования между факультетами института».

Пусть пять факультетов соревнуются по следующим пунктам:

- % успеваемости (отношение числа студентов, сдавших сессию без двоек, к общему числу студентов);
- % качества знаний (сдавшие на 4 и 5 к общему числу студентов);
- участие в студенческих конференциях (0,2 балла за каждого студента);
- число печатных работ (0,5 балла за каждую работу).

Порядок работы.

1. Оформите «шапку» ЭТ.
2. Заполните данными столбцы B, C, D, E, F.
3. В клетку G5 внесите формулу C5/B5 и скопируйте ее в диапазон G5:G9 В клетку H5 внесите D5/B5 и скопируйте в H5:H9. В клетку I5 введите формулу (G5+H5+E5*0.2+F5*0.5) и скопируйте в I5:I9.
4. Построим столбиковую диаграмм) по столбцу «Общий балл». Для этого введем команду /View. Появится основное меню команды:

Show Data Graph-Type Time-Labs Var-Labs Point-Labs Headings Options

Из этого меню производится настройка всех параметров диаграммы Обязательными параметрами являются тип диаграммы (Graph-Type) и данные (Data) Выберите тип диаграммы, войдя в режим Graph-Type и выбрав тип Bar (столбиковая). Определите данные, которые будут выводиться в диаграмме. Для этого войдите в режим Data - появится запрос на ввод первой переменной (Var A). Отметьте диапазон клеток I5:I9. Выведите диаграмму на экран, выбрав пункт Show.

Нажатие клавиш Enter или Esc - возврат в ЭТ.

Без поясняющих меток диаграмма малопонятна. Вновь войдите в меню команды /View и выполните дополнительную настройку параметров:

Headings MainA1	- задание общего заголовка диаграммы из клетки A1;
X-axis A3	- задание заголовка для оси X из клетки A3;
Quit	- возврат в меню команды /View;
Time-Labs A5:A9	- установка меток по оси X;
Var-Labs I5	- задание метки для (первой) переменной.

Выведите диаграмму на экран.

5. Построим столбиковую диаграмму по двум переменным (Var A и Var B). Например, по столбцам B и C.

Определим тип диаграммы и номер:

/View, 2, Graph-Type, Bar

Теперь определим данные, которые будут использоваться в диаграмме. Входим в Data, отмечаем диапазон клеток B5:B9 для первой переменной (Var A), после чего вводим (,), тем самым давая понять, что диаграмма будет строиться по двум переменным. Появляется запрос на ввод 2-й переменной (Var B). Укажите диапазон C5:C9. Выведите диаграмму на экран.

7.3. ЭЛЕКТРОННЫЕ ТАБЛИЦЫ EXCEL

Современные электронные таблицы типа Excel используют манипулятор «мышь»; в них реализован удобный и комфортный интерфейс.

Excel имеет два окна - программное (внешнее) и рабочее (внутреннее). Внутреннее окно Sheet # содержит рабочую страницу (таких страниц несколько, они образуют книгу), представляющую двумерную прямоугольную таблицу (подобную полю SuperCalc). Справа и внизу на рабочей странице расположены линейки со стрелками прокрутки, позволяющие с помощью мыши быстро перемещаться по странице.

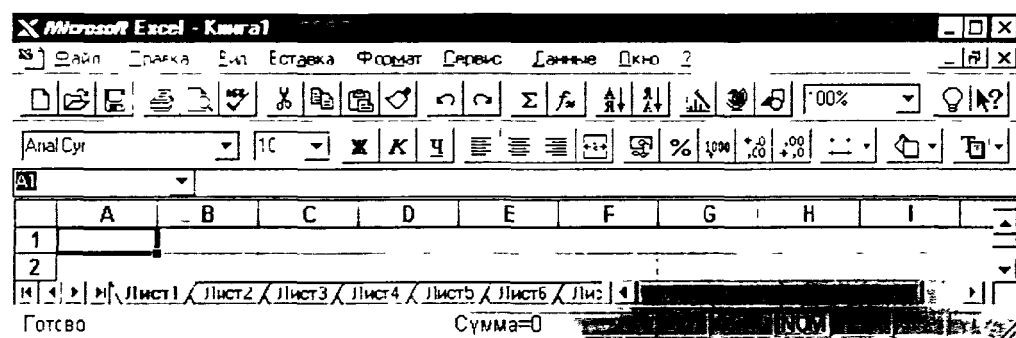


Рис. 2.26. Экран Excel!

В окне Excel (рис. 2.26), как и в других программах пакета MicroSoft Office, под зоной заголовка находится область заголовков меню. Чуть ниже находится основная линейка инструментов.

Кнопки линейки инструментов позволяют быстро и легко вызывать различные функции Excel. Их можно вызывать также через меню.

Если читатель уяснил идеологию электронных таблиц, то он может смело приступать к работе с Excel.

Итак, данные подлежащие обработке размещаются на нескольких листах книги Excel. Введите числа и текст какой-нибудь сметы в ячейки первого листа книги, попытайтесь красиво оформить данные, а затем представьте их в виде диаграммы (попытайтесь!). Далее сохраните данные и напечатайте. У вас должно все получиться!

А теперь опишем несколько правил работы с таблицей. Чтобы выполнить какое-либо действие с данными, помещенными в ячейки (ввод, копирование, удаление, форматирование и т.п.), необходимо их выделить. Чтобы выделить ячейку, укажите на нее и нажмите кнопку мыши. При нажатой кнопке можно выделить диапазон ячеек. После выделения необходимой области нажмите правую кнопку мыши, вызывая контекстное меню, которое позволяет выполнить ряд команд: *Вырезать*, *Копировать*, *Вставить* и т.п.

Изменение данных проводят прямо в ячейке. Перемещение или копирование содержимого ячеек можно осуществить перетаскиванием их с помощью мыши. Чтобы скопировать (а не переместить), держите нажатой клавишу CTRL.

Создание формулы начинается с ввода знака равенства (=). Формула содержит встроенные функции, адреса ячеек, константы. В случае затруднений с формированием формулы используйте *Мастер функций*. Есть великолепная команда *Авто-суммирование*. Выделите столбец или строку данных (и вообще любой диапазон ячеек) и нажмите кнопку «Автосуммирование».

Подобный сервис есть и при оформлении дизайна таблицы. Вручную форматирование проводят стандартным способом. Выделяют ячейку или диапазон, а затем выбирают команды из контекстного меню форматирования. Автоформат позволяет оформить сразу весь текущий лист по шаблону, который выбирается командой *Автоформат* в меню *Формат*.

Перед печатью таблиц (кнопка *Печать*) удобно осуществить предварительный просмотр (соответствующая кнопка в меню *Сохранить*).

Excel работает с несколькими листами книги. Например, на одном листе можно разместить итоговые оценки студенческой группы за пять лет обучения, а на пяти следующих - данные за каждый год обучения. Листы книги могут служить местом для размещения графических иллюстраций, диаграмм.

Для отображения числовых данных в графической форме используют линии, полосы,

столбцы, сектора и другие маркеры, а также их объединенные вариации. Для размещения диаграммы рядом с данными создают *внедренную диаграмму* на том же листе. Можно создать диаграмму на отдельном *листе диаграммы*. Выделите нужный диапазон ячеек, содержащих данные, которые следует представить на диаграмме. Выберите команду *Диаграмма* в меню *Вставка*, а затем команду *На новом листе*. Далее следуйте указаниям *Мастера диаграмм*.

Если нет времени, желания и возможности для относительно сложных построений, используйте автоматическое оформление диаграмм с помощью команды *Автоформат* в меню *Формат*.

Помимо того, что имеется большая встроенная библиотека построения графических образов: графиков, диаграмм, гистограмм. Excel содержит мощный встроенный графический редактор. Принципы работы графического редактора аналогичны подобным системам, описанным в разд. 2.5.

Большое внимание в Excel уделено оформлению книги, ее листов. Для этих целей используют формат текстов и чисел, цвета и заливки, стиль, шрифты и т.д. По сути, графический редактор сопряжен с текстовым процессором, близким по возможностям с издательскими системами типа Word, о котором рассказано в разд. 2.4.

Excel не только «дружен» с текстовыми и графическими системами, но и поддерживает основные действия, характерные для систем управления базами данных (СУБД). В этом смысле современные электронные таблицы (Excel, QuattroPro, Lotus) являются интегрированными программными системами. Более того, у них развит аппарат импортирования и экспортирования данных из других программных систем.

Эти и многие другие приятные сервисные возможности, порой неожиданные, можно для себя открыть при работе с Excel.

Контрольные вопросы и задания

1. Создайте ЭТ «Стипендиальная ведомость».
2. Составьте смету расходов для организации турпохода, если известна общая сумма затрат. Постройте различные виды диаграмм и подготовьте отчет.
3. Составьте компьютерную модель Солнечной системы в электронных таблицах. Считая, что планеты движутся вокруг Солнца по окружностям с постоянной скоростью, определите скорость движения по орбите для каждой планеты. Например, для Марса $V = 2 \cdot 3,14 \cdot 0,387 / 0,24 = 10,1$ (км/ч). Оцените в каких пределах может меняться расстояние от Земли до Марса. При каких расположениях планет достигаются наименьшее и наибольшее значения.
4. Пусть интервал движения автобуса составляет 10 мин. Среднее время ожидания автобуса можно оценить проведя N опытов, разыгрывая случайное число в интервале $[0, 10]$. Найдите среднее значение для серий из 10, 50, 100 опытов. В случае двух маршрутов найдите среднее значение ожидания, когда интервал движения первого автобуса составляет 10 минут, а второго 60 минут. Постройте вычислительную таблицу для трех маршрутов.

§8. ИНТЕГРИРОВАННЫЕ ПРОГРАММНЫЕ СРЕДСТВА

8.1. ПРИНЦИПЫ ПОСТРОЕНИЯ ИНТЕГРИРОВАННЫХ ПРОГРАММНЫХ СИСТЕМ

Программные средства, подробно описанные в предыдущих разделах - системы подготовки текстов и машинной графики, базы данных и электронные таблицы - зачастую не могут удовлетворить запросов пользователей в силу того, что бывает необходимо использовать возможности каждого из них одновременно, в комплексе.

Типичной является ситуация, когда данные, полученные из базы данных, необходимо обрабатывать средствами табличного процессора, представить графически, в виде диаграммы того или иного вида, а затем вставить в текст. Для выполнения работ такого типа существуют, так называемые, интегрированные пакеты - программные средства, совмещающие возможности, характерные в отдельности для текстовых редакторов, графических систем, электронных таблиц, баз данных (и других программных средств). Конечно, такое совмещение возможностей достигается за счет компромисса. Некоторые возможности оказываются в интегрированных пакетах ограниченными

или реализованными не в полной мере. Это касается, в первую очередь, богатства команд обработки базы данных и электронной таблицы, их размеров, макроязыков. Однако преимущества, создаваемые единым интерфейсом объединенных в интегрированном пакете программных средств, неоспоримы. Многие ведущие мировые фирмы, выпускающие программное обеспечение, создали и продолжают развивать свои интегрированные пакеты. Так, фирма «Microsoft» развивает интегрированный пакет Works, известны пакеты Open Access фирмы «Open Access», FrameWork фирмы «Ashton-Tate», Lotus 1-2-3 и Symphony фирмы «Lotus Development Corporation».

Системы Symphony, KnowledgeMan позволяют рассматривать элементы записей в БД, тексты и ячейки электронной таблицы как единое целое: сохраняя на экране одни и те же данные, можно просто как бы менять на них точку зрения, переходя из электронной таблицы в редактор и т.п. В системах предусмотрен многооконный интерфейс: при работе пользователь, переходя из одного окна в другое, меняет «среду», выполняет операции поиска в БД, использует редактор текста и т.д.

Необычно построение системы FrameWork. Все компоненты системы рассматриваются как фреймы - упорядоченные наборы информации, причем пользователь может дробить фреймы на набор фреймов более низкого порядка и т.п. Например, в поле текста можно вставить метку, указывающую на то, что в этом месте должен располагаться участок данных из электронной таблицы, график и пр. Эти фреймы можно будет описать позже, а затем «собрать» все фреймы в единый документ. В системе имеется свой язык программирования Fred.

В интегрированную систему «Мастер» отечественной разработки включены текстовый процессор «Лексикон», процессор электронных таблиц, БД, графический процессор. Кроме того, есть возможность писать программы на языке «Мастер». В этот язык включены функции работы с ячейками электронных таблиц, окнами экрана (рамками), функциями, определенными внутри подсистем. Таким образом, с помощью системы «Мастер» существует возможность сформировать свою версию интегрированной системы, поддерживающую необходимые в данном конкретном приложении функции. Система «Мастер» является примером разумно построенной интегрированной системы, работа с ней естественна и удобна. С помощью этой системы ведется разработка различных информационных систем.

Ограничимся рассмотрением одной из популярных в настоящее время интегрированных систем - Works. По мнению специалистов, этот пакет обладает наиболее наглядным и простым в освоении интерфейсом, полно реализующим основные функции обработки текстов, таблиц и баз данных, а также телекоммуникации по коммутируемым каналам.

8.2. ИНТЕГРИРОВАННЫЙ ПАКЕТ MS-WORKS

После запуска интегрированного пакета MS-Works вы попадаете в интегрированную среду этого программного средства. Опишем некоторые правила работы с составными компонентами пакета.

Работа с текстовым редактором системы MS-Works. Текстовый редактор MS-Works является составной частью интегрированного пакета MS-Works. Чтобы начать работу с текстовым редактором, выполняются следующие действия.

Нажатие на клавишу Alt - переход в главное меню. При этом опция *Файл* оказывается выделенной. Нажатие на клавишу Enter - подтверждение выбора и открытие подменю работы с файлами *Файл*. Если вы создаете новый текст, выберите в этом меню команду *Создать Файл* и нажмите клавишу Enter. В этом случае программа выдаст запрос о том, какой файл следует создать. Выберите из нового открывшегося меню пункт, относящийся к текстовому процессору.

Если текст уже был создан ранее, выберите в меню *Файл* команду *Открыть файл* и введите имя файла или выберите его из списка имеющихся на диске. Выбор файла из списка ничем не отличается от выбора опции из меню.

Для выбора файла можно использовать мышь или клавиши управления курсором, затем нужно нажать клавишу Enter. В этом случае на экране появляется интегрированная среда текстового редактора с уже загруженным текстом.

Подобным образом выбираются и другие команды меню.

Экран текстового редактора разделен на несколько частей. Вверху экрана находится меню. Оно содержит команды, которые используются для редактирования текста и работы с файлами.

Внизу экрана - строка сообщений. Сверху над строкой сообщений - статусная строка. В верхней части экрана под меню имеется линейка. Она определяет границы текста и красную строку. Левая и правая границы текста обозначаются квадратными скобками. Остальная площадь экрана свободна. Это - рабочая область, в которой можно набирать и редактировать текст. На экране постоянно находятся три важных отметки - курсор, буква «п», указывающая начало страницы, и ромбик - метка конца файла. Он не может быть удален с экрана.

При вводе нового текста текстовый редактор сам разместит его в границах, определенных при помощи линейки. Когда текст доходит до конца строки, слово переносится на следующую строку.

Текстовый редактор системы Works использует стандартные для всех редакторов приемы работы с текстом. Если вы недовольны результатами редактирования, изменения легко отменить. Для этого перейдите в меню *Редактирование* (нажмите клавишу Alt, укажите выделенной строкой команду *Редактирование* и нажмите Enter), выберите *Отмена*. Нажмите Enter (или «О»). После этого текст появится на экране в первоначальном виде.

Для того, чтобы сохранить текст на магнитном диске, надо сделать следующее:

- нажать одновременно Alt и Ф (так вы откроете меню работы с файлами *Файл*);
- нажать клавишу С (таким образом вы выберете команду *Сохранить*);
- придумать и ввести имя для вашего файла, в котором будет храниться подготовленный вами текст.

Для выхода из текстового редактора следует нажать Alt и открыть меню работы с файлами *Файл* (нажав клавишу Ф), выбрать команду *Выход* и нажать Enter (можно нажать клавишу X).

Текстовый редактор интегрированной системы Works имеет встроенные средства для проверки правописания - spellинга. Чтобы проверить ошибки в тексте, необходимо поместить курсор в том месте, откуда надо начать проверку, или выделить текст, который надо проверить.

Работа с электронными таблицами системы MS-Works. Электронные таблицы MS-Works являются составной частью интегрированного пакета MS-Works. Для начала сеанса работы с электронной таблицей вызовите MS-Works. Чтобы начать работу с электронными таблицами, выполните следующие действия.

Нажмите клавишу Alt для перехода в главное меню. При этом пункт *Файл* оказывается выделенным. Нажмите клавишу Enter, чтобы открыть меню работы с файлами *Файл*.

Если вы создаете новую таблицу, выберите в этом меню команду *Создать Файл* с помощью клавиш со стрелками. В этом случае программа выдаст запрос о том, какой файл следует создать. Выберите из открывшегося меню пункт, относящийся к электронным таблицам.

Если таблица уже была создана ранее, выберите в меню *Файл* команду *Открыть файл* с помощью клавиш со стрелками и введите имя файла или выберите его из списка имеющихся на диске. Выбор файла из списка ничем не отличается от выбора опции из меню. Для выбора файла вы также используете клавиши со стрелками, затем нажимаете клавишу Enter. В этом случае вы получите на экране электронную таблицу с уже загруженными данными.

Посмотрим, на какие части разделен экран при работе с электронными таблицами. Вверху экрана находится меню. Оно содержит команды, которые используются для работы с электронными таблицами. Для перехода в меню надо нажать клавишу Alt. Строкой ниже находится строка ввода, в которой отображается содержимое активной клетки. Внизу экрана - строка сообщений. Она дает подсказки и описание команд в меню. Сразу над строкой сообщений находится статусная строка, позволяющая следить за ходом работы. Например, она содержит имя активной клетки. Остальная площадь экрана занята электронной таблицей.

После загрузки пустой таблицы на экране видны 7 столбцов (с А до G) и строки с 1-й по 18-ю. Перемещение курсора по таблице осуществляется при помощи клавиш со стрелками. Для того, чтобы ввести в клетку информацию (текст, число, формулу), надо сделать следующее:

- указать клетку таблицы, установив на нее указатель с помощью клавиш управления курсором или мыши;
- набрать с помощью буквенно-цифровых клавиш текст, число, дату, формулу;
- нажать клавишу Enter, чтобы подтвердить ввод данных; если в клетке уже содержатся данные, они будут замещены введенными.

Для отмены ввода информации следует нажать Esc.

С отдельными клетками и с блоками клеток можно выполнять разнообразные операции

(перемещение, копирование, очистку). В этих случаях требуется сначала выделить блок клеток, чтобы произвести с ним какие-то операции. Для того, чтобы выделить целый блок клеток, выполняется следующая последовательность действий:

- курсор таблицы помещается в начало выделяемого блока (его левый верхний угол);
- из меню выбирается команда *Выделить*, а затем *Ячейки* (или нажимается клавиша F8);
- выделенная область расширяется клавишами со стрелками до необходимого размера.

После этого выделенные данные можно изменить, выбрав из меню *Редактирования* одну из команд: *Переместить*, *Копировать*, *Очистить*.

Вычисления в электронных таблицах выполняются с помощью формул. Формула **может** содержать обозначения клеток таблицы, числа, знаки арифметических действий, скобки, определяющие порядок действий, имена функций. Формула начинается со знака равенства (=).

Данные из электронной таблицы могут быть представлены в виде графиков, столбцовых или круговых диаграмм. На диаграммах легче увидеть имеющиеся зависимости между данными. Изменение данных в электронной таблице отражается на диаграмме. Чаще всего используются следующие виды диаграмм:

- линейный график - значения представляются в виде точек, соединенных линиями;
- столбцевая диаграмма - каждое число представлено на диаграмме столбиком.
- круговая диаграмма - значения представляются секторами круга.

Работа с СУБД системы MS-Works. Система управления базами данных MS-Works является составной частью интегрированного пакета MS-Works. Для начала работы с базами данных выполняются следующие действия. Нажатие клавиши Alt - переход в главное меню. При этом опция *Файл* оказывается выделенной. Нажатие клавиши Enter - подтверждение выбора и открытие подменю *Файл* работы с файлами.

Если создается новая база данных, то в этом меню выбирается команда *Создать файл*. В этом случае программа выдаст запрос о том, какой файл следует создать. Из открывшегося меню следует выбрать пункт *Новый файл в формате базы данных*.

Если база данных уже была создана ранее, выберите в меню *Файл* команду *Открыть файл* и введите имя файла или выберите его из списка имеющихся на диске. Выбор файла из списка ничем не отличается от выбора опции из меню. Для выбора файла также используются клавиши со стрелками. После выбора на экране задействована интегрированная среда с уже загруженным файлом базы данных.

Экран при работе с базами данных разделен на зоны. Вверху экрана находится меню. Оно содержит команды, которые используются для работы с базами данных.

Для перехода в меню надо нажать клавишу Alt. Ниже расположена строка ввода, в которой отображается содержимое активного поля. Активным называется поле, в котором находится курсор. Оно выделено цветом. Перемещение курсора по полям осуществляется при помощи клавиш со стрелками.

Внизу экрана находится строка сообщений. Она содержит подсказки и описание команд в меню. Сразу над строкой сообщений - статусная строка. Она позволяет следить за ходом работы. Например, она содержит номер активной записи, имя активного поля этой записи, количество введенных на экран записей из общего числа записей в базе данных.

Остальная площадь экрана свободна. Это рабочая область, в которой происходит работа с базой данных. Для того, чтобы создать базу данных, надо выполнить следующие действия:

- переместить курсор в то место экрана, где должно находиться поле;
- ввести имя поля, после него поставить двоеточие (:);
- нажать клавишу Enter.

Вводятся также значения ширины (число символов) и высоты (число строк) поля. Например, если это поле, в которое будут вводиться фамилии учащихся класса, то его ширина выбирается такой чтобы поместилась самая длинная фамилия, а высота задается в одну строку.

После нажатия клавиши Enter имя поля появится на экране, а справа от имени создается само поле. Задав имена и размеры всех полей базы данных, определим форму базы данных.

После того, как создана форма базы данных, в каждое поле можно занести данные. Содержимым поля может быть текст, число или дата. Можно ввести также формулу. Для того, чтобы ввести данные, необходимо указать курсором нужное поле. Затем можно набирать текст, число

или дату. Нажатие клавиши Enter завершает ввод.

Если необходимо ввести данные и одновременно переместиться в следующее поле, следует нажать Tab.

Если необходимо переместиться в предыдущее поле, нажмите одновременно Shift+Tab. Если после ввода данных в последнее поле записи нажать клавишу Tab, происходит переход к следующей записи. Для перехода к следующей записи также можно использовать сочетание клавиш Ctrl+PgDn.

Для того, чтобы отредактировать поле или имя поля, сделайте следующее:

- укажите курсором нужное поле;
- нажмите клавишу F2, при этом в строке ввода появится содержимое поля;
- внесите необходимые изменения с помощью буквенно-цифровых клавиш, клавиш управления курсором. Back Space и Delete;
- нажмите клавишу Enter.

Все действия с базой данных подобны действиям со специализированными системами управления базами данных.

Одной из наиболее важных возможностей, предоставляемых системой управления базами данных, является поиск записей, удовлетворяющих заданному условию. Для нахождения записи, поля которой содержат заданный текст, выполняются следующие действия:

- переход в меню *Просмотр* (вариант: нажатие Alt и «М»), выбор команды *Таблица*;
- переход в меню *Выделить* (вариант: нажатие Alt и «В») и выбор во вспомогательном меню команды *Поиск*;
- в окно *Что искать* вводится текст, который надо найти.

Часто бывает необходимо выделить из базы данных и наиболее наглядно представить необходимую информацию. Для этого можно создать отчет, сгруппировав данные и подведя итог. Действия при создании отчета таковы:

- переход в меню *Просмотр* (вариант: нажатие Alt и «М»), выполнение команды *Новый Отчет*;
- задание структуры отчета путем внесения необходимых изменений в стандартную форму с помощью буквенно-цифровых клавиш, клавиш управления курсором, Back Space и Delete.

Стандартная форма включает в себя две пустые строки для задания заголовка отчета, две пустые строки для задания заголовков столбцов; ниже располагаются строки записей. В нижней части стандартной формы находятся две пустые строки для вывода итога. В них можно ввести значения стандартных статистических формул СУММ (сумма), СРЧ (среднее значение), МАХ (наибольшее значение), MIN (наименьшее значение).

Если в отчет включаются только записи, удовлетворяющие заданному условию поиска, то после задания структуры отчета выполняются следующие действия:

- возврат в меню *Просмотр* (вариант: нажатие Alt и «М»), выбор команды *Запрос*;
- ввод необходимых условий поиска в появившуюся на экране форму запроса;
- возврат в меню *Просмотр* (вариант: нажатие Alt и «М»), выбор команды *Отчет*.

Иногда бывает нужно вставить информацию из базы данных в документ текстового редактора. Например, если рассылается много одинаковых писем, то адреса можно вставлять из базы данных. Можно вставлять также и отчеты. Для того, чтобы перенести информацию в текст, следует сделать следующее:

- перейти в меню *Просмотр* (вариант: нажать Alt и «М»), выбрать команду *Таблица*;
- перейти в меню *Выделить* (вариант: нажать Alt и «В»), на экране появляется вспомогательное меню (первые четыре строки которого указывают возможные выделяемые области);
- выделить информацию, которую надо скопировать, перейти в меню *Редактирование* (вариант: нажать Alt и «Р») выполнить команду *Копировать*;
- выбрать из открывшегося списка файлов тот, в который необходимо скопировать информацию. После того, как этот файл появился на экране, следует поместить курсор в то место, где надо расположить копию.

Аналогичные действия справедливы при одновременном использовании данных в различных средствах интегрированной системы: текстовом, графическом редакторах, электронных таблицах и СУБД.

1. Что называется интегрированным пакетом?
2. В чем достоинства и недостатки интегрированных пакетов по сравнению со специализированными инструментальными программными средствами?
3. Разработайте информационную систему «Каталог программных продуктов» и подготовьте следующие документы:
 - а) базы данных лицензионных и свободно распространяемых программных средств;
 - б) прайс-лист программных продуктов фирмы «Microsoft»;
 - в) рекламные буклеты по отдельным программным продуктам.

§ 9. ЭКСПЕРТНЫЕ СИСТЕМЫ

Систему искусственного интеллекта, построенную на основе глубоких специальных знаний о некоторой предметной области (полученных от экспертов-специалистов этой области), называют **экспертной системой**. Экспертные системы - один из немногих видов систем искусственного интеллекта (см. гл.1), которые получили широкое распространение и нашли практическое применение. Существуют экспертные системы по военному делу, геологии, инженерному делу, информатике, космической технике, математике, медицине, метеорологии, промышленности, сельскому хозяйству, управлению, физике, химии, электронике, юриспруденции и т.д. И только то, что экспертные системы остаются весьма сложными, дорогими, а главное, узкоспециализированными программами, сдерживает их еще более широкое распространение.

Особенности экспертных систем:

- компетентность - в конкретной предметной области экспертная система должна достигать того же уровня, что и специалисты-люди; при этом она должна пользоваться теми же эвристическими приемами, также глубоко и широко отражать предметную область;
- символьные рассуждения - знания, на которых основана экспертная система, представляют в символьном виде понятия реального мира, рассуждения также происходят в виде преобразовании символьных наборов;
- глубина - экспертиза должна решать серьезные, нетривиальные задачи, отличающиеся сложностью знаний, которые экспертная система использует, или обилием информации; это не позволяет использовать полный перебор вариантов как метод решения задачи и заставляет прибегать к эвристическим, творческим, неформальным методам;
- самосознание - экспертная система должна включать в себя механизм объяснения того, каким образом она приходит к решению задачи.

Экспертные системы создаются для решения разного рода проблем, но они имеют схожую структуру (рис. 2.27); основные типы их деятельности можно сгруппировать в категории, приведенные в табл. 2.5.

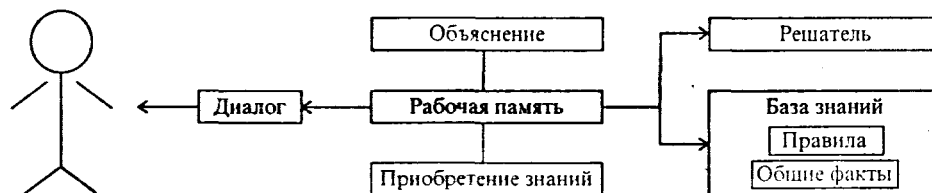


Рис. 2.27. Схема обобщенной экспертной системы

Экспертные системы, выполняющие *интерпретацию*, как правило, используют информацию от датчиков для описания ситуации. Например, это может быть интерпретация показаний измерительных приборов на химическом заводе для определения состояния процесса. Интерпретирующие системы имеют дело не с четкими символьными представлениями проблемной ситуации, а непосредственно с реальными данными. Они сталкиваются с затруднениями, которых нет у систем других типов, потому что им приходится обрабатывать информацию «зашумленную», недостаточную, неполную, ненадежную или ошибочную. Им необходимы специальные методы регистрации характеристик непрерывных потоков данных, сигналов или изображений и методы их символьного представления.

Таблица 2.5 Типичные категории способов применения экспертных систем

Категория	Решаемая проблема
Интерпретация	Описание ситуации по информации, поступающей от датчиков
Прогноз	Определение вероятных последствий заданных ситуаций
Диагностика	Выявление причин неправильного функционирования системы по наблюдениям
Проектирование	Построение конфигурации объектов при заданных ограничениях
Планирование	Определение последовательности действий
Наблюдение	Сравнение результатов наблюдений с ожидаемыми результатами
Отладка	Составление рецептов исправления неправильного функционирования системы
Ремонт	Выполнение последовательности предписанных исправлений
Обучение	Диагностика и исправление поведения обучаемого
Управление	Управление поведением системы как целого

Интерпретирующие экспертные системы могут обработать разнообразные виды данных. Например, система анализа сцен и распознавания речи, используя естественную информацию (в одном случае визуальные образы, в другом - звуковые сигналы), анализирует их характеристики и понимает их смысл. Интерпретация в области химии использует данные дифракции рентгеновских лучей, спектрального анализа или ядерного магнитного резонанса для вывода химической структуры веществ. Интерпретирующая система в геологии использует каротажное зондирование - измерение проводимости горных пород в буровых скважинах и вокруг них, чтобы определить подповерхностные геологические структуры. Медицинские интерпретирующие системы, основываясь на показаниях следящих систем (например, значениях температуры, пульса, кровяного давления), устанавливают диагноз или тяжесть заболевания. В военном деле интерпретирующие системы, получая данные от радаров, радиосвязи и сонарных устройств, оценивают ситуацию и идентифицируют цели.

Экспертные системы, осуществляющие *прогноз*, определяют вероятные последствия заданных ситуаций. Примерами служат прогноз ущерба урожаю от некоторого вида вредных насекомых, оценивание спроса на нефть на мировом рынке, прогнозирование места возникновения следующего вооруженного конфликта на основании данных разведки. Системы прогнозирования иногда используют имитационное моделирование, т.е. программы, которые отражают причинно-следственные взаимосвязи в реальном мире, чтобы сгенерировать ситуации или сценарии, которые могут возникнуть при тех или иных входных данных. Возможные ситуации вместе со знаниями о процессах, порождающих эти ситуации, образуют предпосылки для прогноза. Специалисты по искусственному интеллекту пока что разработали сравнительно мало прогнозирующих систем, возможно потому, что очень трудно взаимодействовать с имитационными моделями и создавать их.

Экспертные системы выполняют *диагностирование*, используя описания ситуаций, характеристики поведения или знания о конструкции компонентов, чтобы установить вероятные причины неправильно функционирующей диагностируемой системы. Примерами служат определение причин заболевания по симптомам, наблюдаемым у пациентов; локализация неисправностей в электронных схемах и определение неисправных компонентов в системе охлаждения ядерных реакторов. Диагностические системы часто являются консультантами, которые не только ставят диагноз, но и помогают в отладке. Они могут взаимодействовать с пользователем, чтобы оказать помощь при поиске неисправностей, а затем предложить порядок действий по их устранению. Медицина представляется вполне естественной областью для диагностирования, и действительно, в медицинской области было разработано больше диагностических систем, чем в любой другой отдельно взятой предметной области. Однако в настоящее время многие диагностические системы разрабатывают для приложений к инженерному делу и компьютерным системам.

Экспертные системы, выполняющие *проектирование*, разрабатывают конфигурации объектов с учетом набора ограничений, присущих проблеме. Примерами могут служить генная инженерия, разработка СБИС и синтез сложных органических молекул.

Экспертные системы, занятые *планированием*, проектируют действия; они определяют полную последовательность действий, прежде чем начнется их выполнение. Примерами могут служить создание плана применения последовательности химических реакций к группам атомов с целью синтеза сложных органических соединений или создание плана воздушного боя с целью нейтрализации определенного фактора боеспособности врага.

Экспертные системы, выполняющие *наблюдение*, сравнивают действительное поведение с ожидаемым поведением системы. Примерами могут служить слежение за показаниями измерительных приборов в ядерных реакторах с целью обнаружения аварийных ситуаций или оценка данных мониторинга больных, помещенных в блоки интенсивной терапии. Наблюдающие экспертные системы сравнивают наблюдаемое поведение с набором допустимых ситуаций нормального поведения. Наблюдающие экспертные системы по самой своей природе должны работать в режиме реального времени и осуществлять зависящую как от времени, так и от контекста интерпретацию поведения наблюдаемого объекта.

Экспертные системы, выполняющие *обучение*, подвергают диагностике, «отладке» и исправлению (коррекции) поведение обучаемого. В качестве примеров приведем обучение студентов отысканию неисправностей в электрических цепях, обучение военных моряков обращению с двигателем на корабле и обучение студентов-медиков выбору антимикробной терапии. Обучающие системы создают модель того, что обучающийся знает и как он эти знания применяет к решению проблемы. Системы диагностируют и указывают обучающемуся его ошибки, анализируя модель и строя планы исправлений указанных ошибок. Они исправляют поведение обучающихся, выполняя эти планы с помощью непосредственных указаний обучающимся.

Экспертные системы, осуществляющие *управление*, адаптивно руководят поведением системы в целом. Примерами служат управление производством и распределением компьютерных систем или контроль за состоянием больных при интенсивной терапии. Управляющие экспертные системы должны включать наблюдающие компоненты, чтобы отслеживать поведение объекта на протяжении времени, но они могут нуждаться и в других компонентах для выполнения любых или всех из уже рассмотренных типов задач: интерпретации, прогнозирования, диагностики, проектирования, планирования, отладки, ремонта и обучения. Типичная комбинация задач состоит из наблюдения, диагностики, отладки, планирования и прогноза.

Рассмотрим примеры наиболее известных классических экспертных систем, с которых началось создание и развитие этого типа программных средств.

MYCIN - это экспертная система, разработанная для медицинской диагностики. В частности, она предназначена для работы в области диагностики и лечения заражения крови и медицинских инфекций. Система ставит соответствующий диагноз, исходя из представленных ей симптомов, и рекомендует курс медикаментозного лечения любой из диагностированных инфекций. Она состоит в общей сложности из 450 правил, разработанных с помощью группы по инфекционным заболеваниям Стэнфордского университета. Ее основополагающим моментом является использование вероятностного подхода.

Система MYCIN справляется с задачей путем назначения показателя определенности каждому из своих 450 правил. Поэтому можно представлять MYCIN как систему, содержащую набор правил вида «ЕСЛИ... , ТО» с определенностью P . В случае MYCIN их предоставили люди-эксперты, которые изложили и правила, и указали свою степень доверия к каждому правилу по шкале от 1 до 10. Установив эти правила и связанные с ними показатели определенности, MYCIN идет по цепочке назад от возможного исхода, чтобы убедиться, можно ли верить такому исходу. Установив все необходимые исходные предпосылки, MYCIN формирует суждение по данному исходу, рассчитанное на основе показателей определенности, связанных со всеми правилами, которые нужно использовать.

Допустим, чтобы получить исход Z , требуется определить предпосылки X и Y , дающие возможность вывести Z . Но правила для определения X и Y могут иметь связанные с ними *Показатели определенности* P и Q . Если значения P и Q были равны 1,0, то исход Z не вызывает сомнения. Если P и Q меньше 1,0 (как это обычно бывает), то исход Z не последует наверняка. Он может получиться лишь с некоторой степенью определенности.

MYCIN не ставит диагноз и не раскрывает его точный *Показатель неопределенности*. Система выдает целый список диагнозов, называя *Показатель определенности* для каждого из них. Все диагнозы с показателями выше определенного, специфического для каждого диагноза уровня,

принимаются как в той или иной степени вероятные, и пользователю вручается список возможных исходов.

Стандартные фразы и грамматические формы были без труда приспособлены к программе, и в результате получился существенно вырожденный диалект английского языка, легко поддающийся программированию. Врачи оказались очень довольными таким результатом, потому что, сами не сознавая того, говорили, используя очень небольшой набор слов английского языка (по крайней мере, когда сообщали о своей работе).

В некотором роде это имеет нечто общее с системой DENDRAL, в которой применяется графический язык, приспособленный к специфической деятельности химиков.

DENDRAL - это старейшая, самая разработанная экспертная система в мире. Или, по крайней мере, старейшая система, названная экспертной.

Химик, приготавливая вещество, часто хочет знать, какова его химическая структура. Для этого существуют различные способы. Во-первых, специалист может сделать определенные умозаключения на основе собственного опыта. Во-вторых, он может исследовать это вещество на спектрометре и, изучая структуру спектральных линий, уточнить свои первоначальные догадки. Во многих случаях это даст ему возможность точно определить структуру вещества. Проблема состоит в том, что все это требует времени и значительной экспертизы со стороны научного сообщества. Здесь-то и оказывается очень полезной система DENDRAL, автоматизирующая процесс определения химической структуры вещества.

В самых общих чертах процесс принятия решения следующий. Пользователь дает системе DENDRAL некоторую информацию о веществе, а также данные спектрометрии (инфракрасной, ядерного магнитного резонанса и масс-спектрометрии), и та, в свою очередь, выдает диагноз в виде соответствующей химической структуры. Можно для простоты представить систему DENDRAL состоящей из двух частей, как если бы в одной экспертной системе были две самостоятельные системы. Первая из них содержит набор правил для выработки возможных химических структур. Вводимая информация состоит из ряда заключений, сделанных химиком, и позволяет судить какие структуры вероятны в том или ином случае.

На выходе первой системы имеется не один простой ответ. Обычно это серия возможных структур - программа не в состоянии точно сказать, какая из них верна. Затем DENDRAL «берет» каждую из этих структур по очереди и использует вторую экспертную систему, чтобы определить для каждой из них, каковы были бы результаты спектрального анализа, если бы это вещество существовало и было на самом деле исследовано на спектрограмме. Процесс, часто именуемый «генерация и проверка», позволяет постоянно сокращать число возможных рассматриваемых вариантов, чтобы в любой момент оно было как можно меньше. В отличие от некоторых экспертных систем DENDRAL задумана не как «игрушка». Она используется не только для проверки теоретических основ экспертных систем, но и реально применяется для определения химических структур.

PROSPECTOR - это экспертная система, применяемая при поиске коммерчески оправданных месторождений полезных ископаемых.

Система PROSPECTOR, по аналогии с MYCIN, содержит большое число правил, относящихся к различным объектам, а также возможных исходов, выведенных на их основе. В этой системе используется также «движение по цепочке назад» и вероятности. Методы этой системы являются одними из лучших среди всех разработанных методов для любой из существующих ныне систем.

Самый простой случай - правила, выражающие логические отношения. Это правила типа «ЕСЛИ X , ТО Z », где событие Z непосредственно вытекает из X . Они остаются такими же простыми, если сопоставить X некоторую вероятность.

Если у X всего один аргумент, то это правило существенно упрощается. Обычно вместо X мы представляем более сложное логическое выражение, например $(X \text{ И } Y)$ или $(X \text{ ИЛИ } Y)$.

Если элементы отношения связаны с помощью логического И и отдельным элементам этого отношения сопоставлены определенные вероятности, то система PROSPECTOR выбирает минимальное из этих значений и присваивает эту минимальную вероятность рассматриваемому возможному исходу. Поэтому когда вероятность наступления события X равна 0,1 и вероятность наступления события Y равна 0,2, то вероятность исхода Z равна 0,1. Легко видеть, почему выбран такой метод: чтобы Z было истинным, и X , и Y должны быть истинными. Это является «жестким» ограничением, поэтому следует брать минимальное значение.

Система PROSPECTOR пользуется методом, основанным на применении формулы Байеса с целью оценки априорной и апостериорной вероятностей какого-либо события. В целом правила в системе PROSPECTOR записываются в виде ЕСЛИ ..., ТО (LS, LN), причем каждое правило устанавливается с отношением правдоподобия как для положительного, так и для отрицательного ответа.

Система PROSPECTOR предлагает пользователю шкалу ответов в диапазоне от -5 до + 5. Нижний предел - это определенно «Да», верхний - определенно «Нет».

Обычно ответ пользователя находится где-то между крайними значениями, и PROSPECTOR корректирует P(H), учитывая LS и LN с помощью линейной интерполяции. Это легко представить себе в виде линейной шкалы, где LN - крайнее левое, а LS - крайнее правое значения.

Кроме экспертных систем MYCIN, DENDRAL и PROSPECTOR существует большое количество других экспертных систем. В табл. 2.6 приводится список некоторых систем, отличительной особенностью которых является наличие большой базы знаний. Этот перечень, конечно, весьма неполный, потому что в последнее время происходит быстрое расширение сферы применения экспертных систем, и полный их перечень был бы огромным и устарел бы почти сразу после его опубликования.

В этом списке приведены также «пустые» экспертные системы (не содержащие конкретных правил предметных областей) и экспертные системы по построению других экспертных систем. Такие системы являются инструментальными средствами для создания прикладных экспертных систем. Они значительно облегчают задачи создания полномасштабной прикладной экспертной системы.

Вообще же инструментальные средства создания экспертных систем (ЭС) классифицируют следующим образом:

- символьные языки программирования, ориентированные на создание ЭС и систем искусственного интеллекта (например, LISP, INTERLISP, SMALLTALK);
- языки инженерии знаний, т.е. языки высокого уровня, ориентированные на построение ЭС (например, OPS-5, LOOPS, Пролог, KES);
- системы, автоматизирующие разработку (проектирование) ЭС (например, KEE, ART, TEIRESLAS, AGE, TIMM); их часто называют окружением (environment) для разработки систем искусственного интеллекта, ориентированных на знания;
- оболочки ЭС (или пустые ЭС) - ЭС, не содержащие знаний ни о какой проблемной области (например, ЭКСПЕРТИЗА, EMYCIN, ЭКО, ЭКСПЕРТ).

Таблица 2.6 Список некоторых экспертных систем

Наименование системы	Назначение системы
MYCIN	Медицинская диагностика
PUFF	Тоже
PIP	»
CASNET	»
INTERNIST	»
SACON	Техническая диагностика
PROSPECTOR	Геологическая диагностика
DENDRAL	Определение химической структуры вещества
SECHS	Тоже
SYNCHM	»
EL	Анализ цепей
MOLGEN	Генетика
MECHO	Механика
PECOS	Программирование
RI	Конфигурирование компьютеров
SU/X	Машинная акустика
VM	Медицинские измерения
SOPHIE	Обучение электронике
GUIDON	Медицинское обучение

TE1RESIAS	Построение базы знаний
EMYCIN	Тоже
EXPERT	»
KAS	»
ROSIE	Построение экспертных систем
AGE	Тоже
HEARSAY III	»
AL/X	»
SAGE	»
Micro-Expert	»

Контрольные вопросы и задания

1. Что отличает экспертные системы от других программ?
2. Какие категории различных типичных проблем решаются экспертными системами?
3. Охарактеризуйте экспертную систему MYCIN.
4. Охарактеризуйте экспертную систему DENDRAL.
5. Охарактеризуйте экспертную систему PROSPECTOR.
6. Какие виды инструментальных программных средств для создания экспертных систем существуют?

§ 10. ИНСТРУМЕНТАЛЬНЫЕ ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ МАТЕМАТИЧЕСКИХ ЗАДАЧ

10.1. НАЗНАЧЕНИЕ ПРОГРАММ

Описанные выше программные системы - текстовые редакторы и издательские системы, электронные таблицы и СУБД - являются инструментальными средствами общего назначения, т.е. могут использоваться для решения наиболее общих задач информационного характера в любой из сфер человеческой деятельности. Вместе с тем в отдельных сферах деятельности часто возникают задачи менее общего характера, такие, например, как проведение математических расчетов типа решения систем уравнений, интегрирования, статистической обработки информации и т.п., которые также требуют использования инструментальных программных средств. Таких более специальных инструментальных программ в настоящее время существует огромное количество. Одно их перечисление заняло бы многие страницы и все равно осталось бы неполным, так как новые «полуприкладные» системы появляются очень часто. Укажем лишь некоторые классы таких инструментальных средств: универсальные математические пакеты, пакеты статистической обработки данных, электронные «органаизеры» - программные средства, облегчающие планирование деятельности, хранение и поиск записей, отслеживающие заданные промежутки времени и т.д.

Ниже коротко описаны две широко используемых как в обучении, так и в практической деятельности системы, предназначенные для решения математических задач — пакет MATHCAD и система аналитических вычислений REDUCE.

10.2. ПАКЕТ MATHCAD

Одним из последних достижений в области инструментальных средств для решения прикладных задач является MATHCAD - физико-математический пакет с включенной в последнюю версию системой искусственного интеллекта SmartMath (разработка NASA), которая позволяет выполнять математические вычисления не только в числовой, но и в аналитической (символьной) форме.

На рисунке 2.28:

1 - палитра операторов;	4-рабочая область;
2 - панель инструментов;	5 - панель форматирования;
3 - главное меню;	6 - строка статуса.

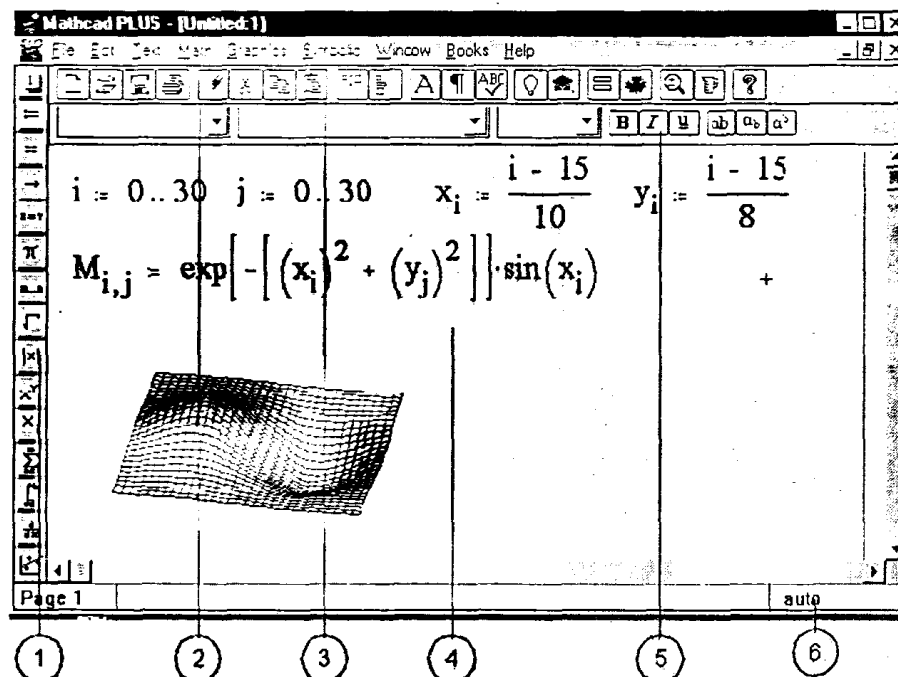


Рис 2.28. Пример экрана MATHCAD

Важное значение разработчики MATHCAD придавали удобству работы с ним и простоте освоения. Интерфейс MATHCAD прост и понятен, полностью отвечает стандартам среды Windows. Все графики и математические объекты могут быть введены щелчком «мыши» с перемещаемых палитр. Обучение пользователя происходит в процессе работы «на ходу» при помощи многочисленных сообщений системы.

Графическая среда MATHCAD позволяет записывать математические формулы в привычном виде, гибко и выразительно представлять данные графически.

Документ MATHCAD состоит из областей различного типа. Текстовые области создаются нажатием кнопки с буквой A на панели инструментов. Математические области возникают, если щелкнуть в свободном месте (появляется красный крестик -визир, фиксирующий место ввода формулы). Области на экране легко можно перетаскивать «мышью» или перемещать командами Cut и Insert меню Edit.

Большинство математических формул записывается в рабочем документе MATHCAD так же, как на листе бумаги. Знаки арифметических операций вводятся с помощью клавиш +, -, *, /.

Для ввода скобок, определяющих порядок выполнения арифметических операций используется клавиша <Space> (пробел). В большинстве случаев система тут же выдает ответ после ввода символа "=" с клавиатуры или с 1-й палитры операторов. В среде MATHCAD знак "=" означает числовой, а знак "стрелка вправо" символьный вывод значения переменной, функции, выражения.

Если последовательно вводить

$$37/5 + 9 = \text{получится } \frac{37}{5+9} = 2,643,$$

$$\text{а если } 37/5 <\text{пробел}> + 9 = \text{получится } \frac{37}{5} + 9 = 16,4.$$

При вводе более сложных операций используют кнопки палитр операторов MATHCAD, находящиеся на экране слева. Для перехода от одной палитры к другой надо щелкнуть на цифре над палитрой.

Стандартные математические функции, такие как cos, sin, arctan, log, exp можно вводить посимвольно, или вставлять из прокручивающегося списка. Чтобы вызвать прокручивающийся список встроенных функций MATHCAD, следует выбрать пункт *Insert Function* из меню *Math*.

Для редактирования выражения надо щелкнуть «мышью» правее элемента выражения, подлежащего изменению, а затем нажать клавишу <Backspace> и ввести нужный элемент. Для немедленного пересчета значения выражения следует щелкнуть «мышью» в стороне от выражения. Все вычисления могут производиться с высокой точностью - число значащих цифр задается из

меню системы и практически не ограничено.

Символ определения «:=» (который можно ввести с 1-й палитры операторов или нажав клавишу «:=») позволяет определять переменные и функции:

$$a := 6 \quad (a - 7) \cdot (a + 2) = -8$$

$$f(x) := \frac{\cos(x)}{a + \frac{x}{2}} \quad f(10) = -0,076$$

Важно следить за тем, чтобы все переменные и функции были определены левее и/или выше тех выражений, где они используются.

Вычислить (протабулировать) функции и выражения для параметров, пробегающих заданный диапазон значений, можно с помощью кнопки "m..n" 1-й палитры. Например, для табуляции функции $f(x)$, приведенной выше, просто вводят $z = f(z)$ и т.д.

$z = 0,5, 2$

z	$F(z)$	$\exp(f(z)) \cdot z$
0	0.167	0
0.5	0.14	0.575
1	0.083	1.087
1.5	0.01	1.516
2	-0.059	1.885

Для создания этих таблиц просто вводят $z = , f(z) =$ и т.д. Имеется ввиду, что функция определена в предыдущем примере (выше).

MATHCAD имеет широкие возможности визуализации числовых данных - 7 видов двумерных и трехмерных графиков. На каждом из двумерных графиков может одновременно находиться до 16 различных кривых, имеющих по 6 атрибутов. Можно создавать собственные библиотеки графических элементов, размещать в рабочем документе MATHCAD произвольные графические изображения.

Для построения графика надо определить с помощью кнопки "m..n" диапазон независимой переменной, а затем создать область графика с помощью кнопки внизу 1-й палитры. После этого вводятся выражения, откладываемые по осям X и Y (в средние поля ввода на соответствующих осях). Для каждой оси может быть введено несколько выражений.

Интегралы и суммы легко вычислять с помощью кнопок 1-й палитры. Для этого достаточно щелкнуть соответствующую кнопку и заполнить появившиеся позиции ввода.

Для выполнения вычислений с матрицами необходимо нажать кнопку с изображением матрицы на 2-й палитре, указать в диалоговом окне число столбцов и строк, нажать кнопку Create и заполнить пустые поля. Теперь, чтобы обратить матрицу A , надо напечатать « A^{-1} », а для вычисления определителя - « A ».

Численное решение уравнения начинается с задания пробного значения корня и требует использования оператора root(,,, ...). Его первый операнд - левая часть уравнения в виде $F(x)=0$, а второй - переменная, по которой ищется корень.

Например,

$$t = 1$$

$$\text{root}(t^2 - \cosh(t), t) = 1,621.$$

MATHCAD корректно оперирует с единицами измерения выводимых числовых результатов и автоматически меняет числовое значение результата при изменении единицы измерения.

Например,

$$\frac{276 \cdot \text{km}}{6 \cdot \text{hr}} = 12.778 \cdot \text{m} \cdot \text{sec}^{-1}.$$

Чтобы пересчитать ответ в других единицах, надо щелкнуть на ответе, затем на втором (дополнительном) поле ввода правее и ввести нужные единицы.

Документ MATHCAD, на котором совмещены текст, графика и формулы, выглядит как страница учебника или научной статьи, при этом формулы являются «живыми» - стоит внести изменения в любую из них, как MATHCAD пересчитает результаты, перерисует графики и т.д. Можно анимировать график, записав его эволюцию при изменяющемся значении параметра, а затем воспроизвести мультипликацию со звуковым сопровождением.

Документы MATHCAD могут быть особым образом «сшиты» в электронные книги. При этом они, сохраняя все свои свойства, оказываются организованными в структуру, обладающую гипертекстовыми ссылками, навигацией, контекстным поиском, открывающимися окнами и т.д.

Доступ к таким электронным книгам может осуществляться по локальным и глобальным сетям - MATHCAD имеет средства для выхода в Интернет и загрузки документов с помощью Интернет-протокола.

При поиске числового результата наряду с общеупотребительными математическими операциями и функциями может быть использовано большое количество встроенных функций, таких как функции отыскания собственных векторов матрицы, решения дифференциального уравнения, генерации последовательности случайных чисел с заданным законом распределения.

В среде MATHCAD имеются функции трех видов: встроенные, пользовательские и вложенные. Это виртуальные функции, производные, интегралы, корни, связанные с соответствующими вычислительными методами и алгоритмами. В меню Symbolic пакета Mathcad PLUS 5.0. включены следующие операции символьной математики:

- вычисление выражения в аналитическом виде;
- вычисление выражения в комплексном виде;
- вычисление числового значения выражения;
- упрощение выражений;
- развертывание выражения;
- разложение на множители;
- группировка выражения;
- вычисление коэффициентов полинома;
- поиск производной по переменной;
- интегрирование по переменной;
- решение уравнения в аналитическом виде;
- подстановка в выражение;
- разложение в ряд;
- представление в виде смешанной дроби;
- транспонирование матрицы;
- инвертирование (обращение) матрицы;
- нахождение детерминанта (определителя) матрицы;
- преобразование Фурье;
- обратное преобразование Фурье;
- преобразование Лапласа;
- обратное преобразование Лапласа;
- Z-транспонирование;
- обратное Z-транспонирование;
- пределы (команд нет.есть кнопки-иконки).

В системе имеются разнообразные способы ввода числовых данных: с клавиатуры, из других приложений, например, электронных таблиц, с использованием технологии OLE или DDE или буфера обмена, непосредственно их файлов, с использованием разнообразных функций файлового доступа.

Интеллектуальная система SmartMath включается в работу двумя способами: одноименной командой из меню Math или нажатием и «притопплением» на панели инструментов кнопки-иконки с изображением кафедралки - головного убора средневековых ученых.

SmartMath позволяет работать не только в ручном, но и в автоматическом режиме. Режим автоматических символьных преобразований включается опцией *Live Variable* в меню *Math*.

Искомое выражение появляется правее, ниже или вместо исходного, заданного пользователем. Место для результата задается установкой *Derivation Format...* и *Derive in Place* в меню *Symbolic*. Если в исходном выражении пользователь что-то поменяет, то ему придется все действия повторить, не забыв при этом стереть предыдущий вариант ответа. В автоматическом режиме за исходным выражением нажатием соответствующей кнопки-иконки ставится знак «стрелка вправо». А чтобы система SmartMath поняла, в каком направлении необходимо вести преобразования, введены 7 ключевых слов: *factor*, *expand*, *series*, *simplify*, *complex*, *float* и *assume*. Эти слова можно считать зачатками нового языка программирования, ориентированного не на вычислительный, а на аналитический процесс.

Второй режим системы SmartMath связан с оптимизацией численных расчетов.

Ключевое слово *optimize*, поставленное перед суммой (произведением, интегралом, пределом), заставляет систему SmartMath отойти от лобовой атаки. Если оптимальное решение найдено, то правее выражения появляется красная шестиугольная звезда.

Пользователь может просмотреть не только численный результат, но и аналитическое выражение, упростившее расчеты. Оно заносится в специальный буфер, отображенный на диске командой *Show SmartMath...* меню *Math* или щелчком по красной звездочке. Оптимизационное выражение можно записать в переменную, которая будет уже иметь, не числовой, а символьный тип. Оптимизировать расчеты можно и без слова *optimize*, включив опцию *Optimize* в меню *Math*. В этом случае оптимизироваться будут все выражения без особого на то приглашения.

Оптимизация не только ускоряет расчеты, но и повышает их точность. И не только количественно, но и качественно за счет исправления методологических ошибок (промахов) численных методов.

10.3. СИСТЕМА АНАЛИТИЧЕСКИХ ПРЕОБРАЗОВАНИЙ REDUCE

Развитие вычислительной техники начиналось с автоматизации выполнения арифметических действий. Вместе с тем известно, что компьютеры могут успешно оперировать математическими символами. Область вычислительной математики, связанная с аналитическими преобразованиями и получившая название компьютерной алгебры, в настоящее время развивается и получает широкое распространение в различных направлениях науки и образования. Основным объектом, над которым производит действие компьютер, является аналитическое (символьное) выражение, организованное и преобразуемое по заданным логическим правилам. Сегодня возможно компьютерное интегрирование и дифференцирование символьных выражений, перестановки и перегруппировки членов, приведение подобных членов, подстановки в выражения с последующим их преобразованием.

Очевидно, что известные системы программирования (Паскаль, СИ, Бейсик и т.п.) мало пригодны для анализа и преобразования символьной информации. Для этих целей созданы и развиваются специальные системы аналитических преобразований, которые можно разделить на универсальные, специализированные и общего назначения. Наибольшей популярностью пользуется универсальная система символьных вычислений REDUCE, автором которой является профессор А.Хиен. Система Reduce написана на языке высокого уровня ЛИСП.

Язык Reduce, составляющий ядро системы, трансляторы для которого разработаны для всех распространенных типов ЭВМ, предназначен прежде всего для проведения вычислений в аналитическом виде. Язык «знает» все операции алгебры с многочленами, приведением подобных членов, раскрытием скобок; все базовые элементарные функции, в том числе и в комплексной форме, ряд других функции; широкий набор операции над матрицами, включающий как входящие в обычные вузовские программы, так и выходящие за их пределы (например, функции от матриц); очень хорошо дифференцирует и несколько хуже вычисляет первообразные (но все же значительно лучше, чем большинство студентов, изучивших стандартный курс интегрального исчисления); умеет делать и ряд других действий.

Насколько это существенно для решения некоторых задач, показывает следующий пример. В одной из диссертаций по физике магнитных явлений диссертант (дело было в середине 50-х годов) потратил несколько лет на решение в принципе несложной задачи, требовавшей, однако, про-

ведения совершенно фантастического (для человека) количества операторных коммутаций и последующего приведения подобных членов. Каждое отдельное неразрывное преобразование требовало, по-видимому, нескольких месяцев неустанной работы при максимальной аккуратности и напряжении. Поручить же эту работу ЭВМ было невозможно, так как в те времена языков аналитических преобразований в практически пригодном виде не существовало. О независимой проверке работы не приходилось и думать - это потребовало бы от другого человека не менее года работы. Примерно через 15 лет все эти выкладки были проделаны на ЭВМ в системе Reduce за несколько дней; оказалось, что автор диссертации почти все выкладки сделал безошибочно.

Разумеется, язык «умеет» производить и численные операции, причем его арифметика имеет произвольную точность, не привязанную к способу представления чисел с плавающей запятой в регистрах процессора и ячейках ОЗУ. Получить при вычислении 20 или 50 значащих цифр в результате для Reduce вполне возможно.

В системе Reduce программа записывается и выполняется по предложениям, каждое из которых представляет собой последовательность символов. Предложение завершается одним из символов: ; (точка с запятой), п (кружок с черточками). Если предложение оканчивается знаком ";", то результат его выполнения выводится на экран дисплея или печатающее устройство. В случае знака п вывод не происходит.

В системе Reduce каждая переменная имеет имя и значение. Если переменной не присвоено какое-либо значение, то имя переменной является ее значением. В этом заключается одно из принципиальных отличий подобных систем от традиционных языков программирования. Первоначально имя и значение переменной совпадают между собой, и такая переменная называется свободной.

Запуск программы на выполнение в системе осуществляется клавишей <Enter>.

Ниже приведем несколько примеров программ в системе Reduce, которые позволят получить первоначальные представления о системах аналитических преобразований символьной информации.

Пример 1.

A; XI; SS# ABCDIF; <Enter>

Листинг результата программы следующий:

A
XI
ABCDIF

Комментарий: все переменные являются свободными, т.е. их значения совпадают с именами.

Пример 2.

A:=123456789# B:= 123456789123456789#A*B; <Enter>

Листинг результата программы следующий:

15241578765432099750190521

Комментарий: переменным A и B присваиваются целочисленные значения и вычисляется их произведение, причем результат вычисления точный без округления.

Пример 3.

A:=S# A; A:=X*Y# A; Q:=X:=Y# Q; X; <Enter>

Листинг результата программы следующий:

S
X*Y
Y
Y

Комментарий: переменной A присваивается сначала значение S, затем - X*Y.

Пример 4.

13; 3+6; 2**64; 11-20; 25/(-125); 2*(3*A-6)/6; <Enter>

Листинг результата программы следующий:

139 18446744073709551616 (-9) (-1)/5A-2

Комментарий: при проведении алгебраических преобразований для записи сложных выражений используют имена переменных и знаки арифметических операций.

Пример 5.

I**2; A:=X+I*Y# B:=X-I*Y# A*B; A**2; <Enter>

Листинг результата программы следующий:

-1
X-52-0+Y-52
' 2*I*X*Y + X-52-0-Y-52

Комментарий: для использования комплексных чисел за латинской буквой I закреплено значение мнимой единицы.

Пример 6.

OPERATOR F,W; W(X); F(5*X); (F(X)+A)**2; <Enter>

Листинг результата программы следующий:

W(X) F(5*X) F(X)-52-0 + 2*A*-F(X) + A-52

Комментарий: имена в скобках используют для обозначения операторов или функций, которые описываются предварительно командой OPERATOR.

Пример 7.

DF(X**2,X); DF(Y,Y,2); DF(X**3*Y*82*Z**3,X,3,Y,Z,2); DF(Y,X); <Enter>

Листинг результата программы следующий: 2*X 0 72*Y*Z O

Комментарий: встроенный оператор DF используется для вычисления частных производных по отношению к одной или нескольким переменным, первым аргументом в скобках является дифференцируемое выражение, далее - аргументы, по которым проводится дифференцирование, и числа, указывающие порядок производной.

Пример 8.

INT(X**2,X); INT(SIN(X),X); <Enter>

Листинг результата программы следующий:

X-53-0/3 - COS(X)

Комментарий: оператор INT используется для вычисления интегралов, на первом месте стоит интегрируемое алгебраическое выражение, на втором месте указывается переменная интегрирования.

Мы привели наиболее простые возможности системы Reduce. Подробнее ознакомиться с работой подобных систем читателю рекомендуется по специальным учебным пособиям и монографиям.

Контрольные вопросы

1. В чем основные отличия переменных в традиционных системах программирования от систем аналитических преобразований типа REDUCE?
2. В каких задачах предпочтительнее использовать методы компьютерной алгебры?

§ 11. КОМПЬЮТЕРНОЕ ТЕСТИРОВАНИЕ

11.1. ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ КОМПЬЮТЕРНЫХ ТЕСТОВ ПРЕДМЕТНОЙ ОБЛАСТИ

Компьютеры в обучении - вопрос, требующий отдельного рассмотрения. Отметим лишь, что различные варианты АОС (автоматизированных обучающих систем) вобрала в себя лучшие достижения компьютерных технологий и стали широко популярными не только в учебных заведениях, но и при подготовке персонала в промышленности, различных социальных сферах, военном деле и т.д.

Обучение - многогранный процесс, и контроль знаний - лишь одна из его сторон. Однако именно в ней компьютерные технологии продвинулись максимально далеко, и среди них тестирование занимает ведущую роль. В ряде стран тестирование потеснило традиционные формы контроля - устные и письменные экзамены и собеседования.

По-видимому, многие преподаватели уже прошли через некоторую эйфорию при создании тестов и поняли, что это - весьма непростое дело. Куча бессистемно надерганных вопросов и ответов - далеко еще не тест. Оказывается, что для создания адекватного и эффективного теста нужно затратить много труда. Компьютер может оказать в этом деле немалую помощь.

Существует специальная *теория тестирования*, оперирующая понятиями *надежность*, *валидность*, *матрица покрытия* и т.д., не специфических именно для компьютерных тестов. Здесь мы не будем в нее углубляться, сосредоточившись в основном на технологических аспектах.

Широкое распространение в настоящее время получают инструментальные авторские системы по созданию педагогических средств: обучающих программ, электронных учебников, компьютерных тестов. Особую актуальность для преподавателей школ и вузов приобретают программы для создания компьютерных тестов - тестовые оболочки. Подобных программных средств существует множество и программисты-разработчики готовы строить новые варианты, так называемых, авторских систем. Однако широкое распространение этих программных средств сдерживается отсутствием простых и нетрудоемких методик составления тестовых заданий, с помощью которых можно «начинять» оболочки. В настоящем разделе представлены некоторые подходы к разработке компьютерных тестов.

Проектирование модели знаний. Эксперты чаще используют метод нисходящего проектирования модели знаний (технология «сверху - вниз»). Вначале строится генеральное содержание предметной области с разбивкой на укрупненные модули (разделы). Затем проводится детализация модулей на элементарные подмодули, которые, в свою очередь, наполняются педагогическим содержанием.

Другой метод проектирования «снизу - вверх» (от частного к общему) в большинстве случаев реализуется группой экспертов для разработки модели знаний сложной и объемной предметной области, или для нескольких, близких по структуре и содержанию, предметных областей.

Каждый модуль предполагает входящую информацию, состоящую из набора необходимых понятий из других модулей и предметных областей, а на выходе создает совокупность новых понятий, знаний, описанных в данном модуле.

Модуль может содержать подмодули. Элементарный подмодуль - неделимый элемент знания - может быть представлен в виде базы данных, базы знаний, информационной модели. Понятия и отношения между ними представляют семантический граф.

Приведем пример элемента модуля знаний по теме «Исследование графиков функций», рис. 2.29:

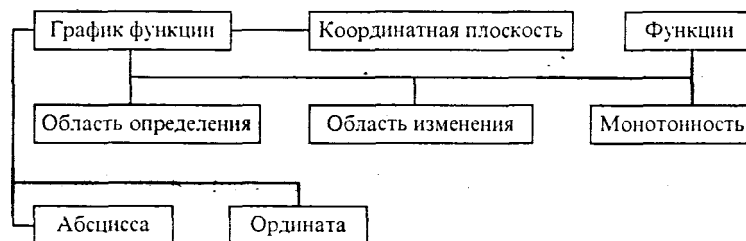


Рис. 2.29. Пример элемента модуля знаний

Модульное представление знаний помогает:

- организовать четкую систему контроля с помощью компьютерного тестирования, поскольку допускает промежуточный контроль (тестирование) каждого модуля, итоговый по всем модулям и их взаимосвязям, а также эффективно использовать методику «черного ящика»;
- осуществлять наполнение каждого модуля педагогическим содержанием;
- выявить и учитывать семантические связи модулей и их отношения с другими предметными областями.

Этапы разработки компьютерных тестов. Можно выделить два принципиальных способа контроля (тестирования) некоторой системы:

- 1) метод «белого ящика» - принцип тестирования экспертной модели знаний;
- 2) метод «черного ящика» - тестирование некоторой сложной системы по принципу контроля входных и выходных данных (наиболее подходит для компьютерного тестирования).

Введем ряд определений и понятий.

Тестирование - процесс оценки соответствия личностной модели знаний ученика экспертной модели знаний. Главная цель тестирования - обнаружение несоответствия этих моделей (а не измерение уровня знаний), оценка уровня их несоответствия.

Тестирование проводится с помощью специальных тестов, состоящих из заданного набора тестовых заданий.

Тестовое задание - четкое и ясное задание по предметной области, требующее однозначного ответа или выполнения определенного алгоритма действий.

Тест - набор взаимосвязанных тестовых заданий, позволяющих оценить соответствие знания ученика экспертной модели знаний предметной области.

Тестовое пространство - множество тестовых заданий по всем модулям экспертной модели знаний.

Класс эквивалентности - множество тестовых заданий, таких, что выполнение одного из них учеником гарантирует выполнение других.

Полный тест - подмножество тестового пространства, обеспечивающего объективную оценку соответствия между личностной моделью и экспертной моделью знаний.

Эффективный тест - оптимальный по объему полный тест.

Самой сложной задачей эксперта по контролю является задача разработки тестов, которые позволяют максимально объективно оценить уровень соответствия или несоответствия личностной модели знания ученика и экспертной модели.

Подбор тестовых заданий осуществляется экспертами-педагогами методологией «белого ящика», а их пригодность оценивают с помощью «черного ящика», рис. 2.30.

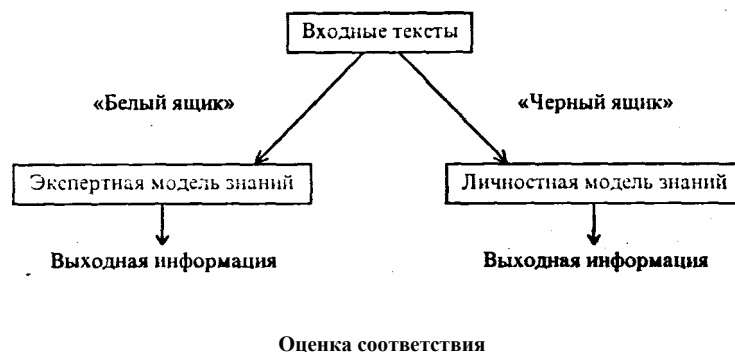


Рис. 2.30. Схема создания тестовых заданий

Самый простой способ составления тестовых заданий - формирование вопросов к понятиям, составляющим узлы семантического графа, разработка упражнений, требующих для выполнения знания свойств выбранного понятия. Более сложным этапом является разработка тестовых заданий, определяющих отношения между понятиями. Еще более глубокий уровень заданий связан с подбором тестов, выявляющих связь понятий между отдельными модулями.

Множество тестовых заданий (тестовое пространство), вообще говоря, согласно принципу исчерпывающего тестирования, может быть бесконечным. Однако в каждом реальном случае существует конечное подмножество тестовых заданий, использование которых позволяет с большой вероятностной точностью оценить соответствие знаний ученика заданным критериям по экспертной модели знаний (полный тест).

Из полного теста можно выделить эффективный тест (оптимальный по объему набор тестовых заданий, гарантирующий оценку личностной модели ученика заданным критериям). Выбор эффективного теста зависит от удачного разбиения тестового пространства на классы эквивалентности, пограничные условия, создания тестов на покрытие путей и логических связей между понятиями и модулями.

В дальнейшем необходим тестовый эксперимент на группе учащихся, который позволит провести корректировку и доводку теста до вида эксплуатации (методика черного ящика).

Таким образом, построение компьютерных тестов можно осуществлять в следующей последовательности:

- формализация экспертной целевой модели знаний;
- нисходящее (или снизу - вверх) проектирование тестового пространства;
- формирование и наполнение тестовых заданий;
- формирование полного компьютерного теста;
- тестовый эксперимент;
- выбор эффективного теста;
- анализ, корректировка и доводка теста до вида эксплуатации.

11.2. ТИПЫ КОМПЬЮТЕРНЫХ ТЕСТОВ

В соответствии с моделью знаний, выделим три класса компьютерных тестов на знания, умения и навыки. Отметим, что типы компьютерных тестовых заданий определяются способами однозначного распознавания ответных действий тестируемого.

1. Типы тестовых заданий по блоку «знания» — вопросы альтернативные (требуют ответа да - нет);

- вопросы с выбором (ответ из набора вариантов);
- вопросы информативные на знание фактов (где, когда, сколько);
- вопросы на знание фактов, имеющих формализованную структуру (в виде информационной модели или схемы знаний);
- вопросы по темам, где имеются однозначные общепринятые знаковые модели: математические формулы, законы, предикатные представления, таблицы;
- вопросы, ответы на которые можно контролировать по набору ключевых слов;

- вопросы, ответы на которые можно распознавать каким-либо методом однозначно.

2. Типы тестовых заданий по блоку «навыки» (распознавание деятельности: манипуляции с клавиатурой; по конечному результату):

- задания на стандартные алгоритмы (альтернативные да - нет, выбор из набора вариантов);
- выполнение действия.

3. Типы тестовых заданий по блоку «умения». Те же самые, что для навыков, но используют нестандартные алгоритмы и задачи предметной области при контроле времени их решения:

- задания на нестандартные алгоритмы (альтернативные да - нет, выбор из набора вариантов);
- выполнение действия.

Выбор типов тестов определяется:

- особенностями инструментальных тестовых программ (тестовыми оболочками);
- особенностями предметной области;
- опытом и мастерством экспертов.

11.3. ИНСТРУМЕНТАЛЬНЫЕ ТЕСТОВЫЕ ОБОЛОЧКИ

Для создания тестов по предметной области разрабатываются специальные инструментальные программы-оболочки, позволяющие создавать компьютерные тесты путем формирования базы данных из набора тестовых заданий.

Инструментальные программы, позволяющие разрабатывать компьютерные тесты, можно разделить на два класса: универсальные и специализированные. Универсальные программы содержат тестовую оболочку как составную часть. Среди них Адонис (Москва), Linkway (Microsoft), Фея (Томск), Радуга (Москва) и т.п. Специализированные тестовые оболочки предназначены лишь для формирования тестов. Это - Аист (Москва), I_pow (Иркутск), Тест (Красноярск) и др.

Для того, чтобы разработать компьютерный вариант теста с помощью одной из названных выше программ, необходимо уяснить, какие формы тестовых заданий они допускают.

Как правило, компьютерные формы представления тестовых заданий могут выглядеть следующим образом.

1. Вопросы с фасетом. Задание вопроса, в котором меняются признаки.

Пример.

Назовите столицу страны АНГЛИЯ : ? _____.

2. Вопросы с шаблоном ответа.

Пример.

В каком году произошла Октябрьская революция? В ____ году.

3. Вопросы с набором ключевых слов (изображений, обозначений), из которых можно конструировать ответ.

Пример.

Какие силы действуют на тело, движущееся по наклонной плоскости? (сила трения, сила упругости, сила тяжести, сила реакции опоры).

4. Закрытая форма вопроса: номер правильного ответа.

Пример.

Какой климат в Красноярском крае?

1. Континентальный.
2. Субтропики.
3. Умеренный.
4. Резко-континентальный.
5. Задание на соответствие: несколько вопросов и несколько ответов.

Пример.

а) Кто автор планетарной модели?	а) Лермонтов М.Ю.
б) Кто автор закона тяготения?	б) Резерфорд
в) Кто автор поэмы «Мцыри»?	в) Ньютон

6. Конструирование ответа (шаблонный и безшаблонный варианты): ответ формируется путем последовательного выбора элементов из инструментария по типу меню.

Пример.

Чему равна производная функции $y = \sin(x) + \cos(x)$?

$y' = (\sin(x), \cos(x), \operatorname{tg}(x), +, -, /, *, \log(x), 1, 2, 3, 5 \text{ и т.д.})$

7. Задание на конструирование изображений: с помощью графредактора, меню изображения (аналогично предыдущему примеру).

8. Задание на демонстрацию с движущимися объектами. Ответ в виде действий тестируемого (определенный набор клавиш).

Пример.

Клавиатурный тренажер на время.

Перечисленные формы компьютерного представления тестовых заданий не исчерпывают их многообразия. Многое зависит от мастерства и изобретательности эксперта по тестированию. При создании тестов важно учитывать многие обстоятельства, например, личность тестируемого, вид контроля, методику использования тестов в учебном процессе и т.п.

Хорошим считается тест если

- он восприимчив к угадыванию тестируемым;
- он восприимчив к невнимательности и ошибочным действиям тестируемого;
- он положительно влияет на тестируемого и педагога.

При этом тест используется обучаемым как тренажер и орудие самоконтроля. Для учителя тест служит для корректировки учебного процесса, используется как вспомогательное средство текущего контроля знаний, как дидактические средства обучения, для дистанционного обучения,

11.4. ПРИМЕР ТЕСТА ПО ШКОЛЬНОМУ КУРСУ ИНФОРМАТИКИ

В 1996 г. Республиканский центр тестирования использовал тесты по некоторым школьным предметам, в частности по информатике. Ниже приводится один из его вариантов (разработчики: Н.Г.Граве, И.А.Елисеев, Г.В.Тюрникова). Тесты построены на основе канонического принципа: вопрос и варианты ответа.

Разработчиками выбрана следующая модель знаний школьного курса информатики:

N Модуль 1. **Введение**

1. Измерение информации ,
2. Свойство информации
3. Измерение информации
4. Предмет информации. Фундаментальные понятия
5. История развития вычислительной техники

Модуль 2. **Устройство и работа ЭВМ**

6. Состав информационно-измерительного комплекса
7. Поколения ЭВМ
8. Арифметические основы ЭВМ
9. Состав информационно-измерительного комплекса
10. Арифметические основы ЭВМ
11. Физические основы ЭВМ
12. Состав информационно-измерительного комплекса

Модуль 3. **Алгоритмизация**

13. Величины, тип, имя, значения, вид
14. Величины, тип, имя, значения, вид
15. Величины, тип, имя, значения, вид
16. Типы алгоритма
17. Способы описания
18. Способы описания
19. Алгоритм, свойства
- 20-24. Остальные вопросы как единый подраздел

Модуль 4. **Информационные технологии**

- 25-28. Операционные системы

- 29-30. Текстовый, графический, музыкальный редакторы
31-32. Базы данных
33. Электронные таблицы
Модуль 5. **Заключение**
34-36. Перспективы развития

РОССИЙСКИЙ ТЕСТ ПО ИНФОРМАТИКЕ №01

01. Килобайт - это
1) 1000 символов; 2) 1024 байт; 3) 8 бит; 4) 1000 байт.
02. Достоверность - это свойство
1) алгоритма; 2) компьютера; 3) информации; 4) языка программирования.
03. Наибольший объем памяти требуется для хранения
1) "10"; 2) 10; 3) "десять"; 4) (10).
04. Носителем информации является
1) провода; 2) принтер; 3) классный журнал; 4) телефон.
05. Первая машина, автоматически выполняющая все 10 команд, была
1) машина С.А.Лебедева; 2) машина Ч.Бэббиджа; 3) абак; 4) Pentium.
06. Минимально необходимый набор устройств для работы компьютера содержит
1) принтер, системный блок, клавиатуру; 2) процессор, ОЗУ, монитор, клавиатуру; 3) монитор, винчестер, клавиатуру, процессор; 4) системный блок, дисководы, мышь.
07. Элементарной базой ЭВМ третьего поколения являются
1) ЭЛТ (электронно-лучевая трубка); 2) светодиоды;
3) ИС (интегральные схемы); 4) транзисторы.
08. Число 3210-это
1) 1000002; 2) 358; 3) 2116; 4) 100001.
09. К внешним запоминающим устройствам относится
1) процессор; 2) дискета; 3) монитор; 4) жесткий диск.
10. Определить сумму трех чисел: $0012 + 0178 + 1112$
1) 02310; 2) 00910; 3) 1112; 4) 10002.
11. Перевести число 3210 в двоичную систему счисления
1) 100000; 2) 111111; 3) 101010; 4) 100001.
12. К внутренним запоминающим устройствам относится
1) монитор; 2) жесткий диск; 3) оперативная память (RAM); 4) флоппи-диск.
13. Неверно записанное выражение
1) $+3$; 2) $\text{tg}(+3)$; 3) $\text{tg}(-3)+1$; 4) $-\sin(-3)+(1)*(\text{tg}(+1))$.
14. По выполнении следующего алгоритма
 $x:=7$; $y:=12+5$; $y:=y+y-x$
значение x будет
1) 7; 2) 89; 3) 94; 4) 47.
15. Если исполнить
 $X:=2$; $Y:=X+3$; $X:=X+1$; $Y:=X+3*Y$,
то значение Y равно
1) 0; 2) -10; 3) 18; 4) 6,5.
16. При $t > 17$ будет ложно
1) $t=17,01$; 2) $t>212$ и $t<1000$; 3) $t=17$; 4) $t>17$ и $t<20$.
17. До какого числа должно измениться значение счетчика i в фрагменте алгоритма
 $a:=1$
нц для i от 2 до $\langle \dots \rangle$
 $a:=a*i$
 $i:=i+1$ $a:=a*i$
кц
чтобы a стало равно 11?
1) 8; 2) 9; 3) 10; 4) 11.

18. Для вывода данных в блок-схемах используют фигуру



19. Геометрическая фигура используется в блок-схемах для обозначения

1) условия; 2) останова; 3) любого действия; 4) цикла «для».

20. Не является свойством алгоритма

1) универсальность; 2) массовость; 3) результативность; 4) дискретность.

21. При составлении алгоритма для вычисления функции $y=a*\sin(x)$ аргументами являются

1) $\sin$; 2) a, x ; 3) x ; 4) x, y .

22. Сколько раз выполнится цикл

```
i-1;  
a:=10;  
n:=2;  
нц пока a>0  
a:= a-n*i  
кц  
1)0; 2) 10; 3)5; 4) 4.
```

23. В качестве имени переменной может быть

1) 1996; 2) a1996; 3) 1996a; 4) -1996.

24. Для описания циклического алгоритма используется конструкция

1) ПОКА; 2) ЕСЛИ; 3) ВЫБОР; 4) ПРОЦЕДУРА.

25. Какая программа является интерпретатором команд MS DOS?

1) AUTOEXEC.BAT; 2) MSDOS.SYS; 3) CONFIG.SYS; 4) COMMAND.COM

26. Минимально необходимый набор файлов для работы компьютера в MSDOS

1) IO.SYS, MSDOS.SYS; 2) IO.SYS, MSDOS.SYS, COMMAND.COM;
3) IO.SYS, MSDOS.SYS, COMMAND.COM, CONFIG.SYS;
4) IO.SYS, MSDOS.SYS, COMMAND.COM, AUTOEXEC.BAT.

27. Сколько символов в своем полном имени может содержать директория?

1) 11; 2) 8; 3) 7; 4) 12.

28. Неверным будет утверждение

1) файл с расширением .TXT может быть не текстовым;
2) системный диск может не содержать файл CONFIG.SYS;
3) файл AUTOEXEC.BAT может не содержать ни одной строки (ни одного байта);
4) файл должен содержать в расширении не менее трех букв.

29. Текстовый редактор Лексикон - это

1) прикладная программа; 2) базовое программное обеспечение;
3) сервисная программа; 4) редактор шрифтов?

30. Под термином «интерфейс» понимается

1) внешний вид программной среды, служащий для обеспечения диалога с пользователем;

2) связь текстового редактора с устройством печати;
3) совокупность файлов, содержащихся в одном каталоге;
4) устройство хранения графической информации.

31. База данных - это 1) текстовый редактор; 2) совокупность связанных между собой сведений; 3) операционная оболочка; 4) утилиты NC?

32. Графический редактор нужен для

1) нормальной работы баз данных; 2) быстрого поиска информации;
3) проигрывания звуковых файлов; 4) создания рисунков.

33. В отличие от бумажных табличных документов, электронные таблицы обычно

1) имеют большую размерность; 2) позволяют быстрее производить расчеты;
3) обладают всеми свойствами, перечисленными в пунктах 1 -2;
4) стоят дороже.

34. Что делает невозможным подключение компьютера к глобальной сети?

- 1) тип компьютера; 2) состав периферийных устройств;
3) отсутствие винчестера; 4) отсутствие телефона.
35. Дан E-mail: artem@wremtech.msk.ru. Слово msk означает
1) город назначения; 2) тип компьютера; 3) каталог; 4) имя пользователя.
36. Первый PHOTO CD был произведен фирмой
1) IBM; 2) APPLE; 3) KODAK; 4) POLAROID.

Ответы на тестовые задания

01 - 2); 02 - 3); 03 - 3); 04 - 3); 05 - 2); 06 - 2); 07 - 3); 08 - 4); 09 - 2); 10 - 4); 11 - 4);
12 - 2); 13 - 4); 14 - 1); 15 - 3); 16-3); 17 - 4); 18 - 3); 19 - 2); 20 - 1); 21 - 3); 22 - 3); 23 - 2); 24 -
1); 25 - 4); 26 - 2); 27 - 2); 28 - 4); 29 - 1); 30 - 1); 31- 2); 32 - 4); 33 - 3); 34 - 4); 35-1); 36-1).

Контрольные вопросы и задания

1. Разработайте модель знания по школьному разделу «действия с дробями», используя модульный принцип.
2. Разработайте тест на знание таблицы умножения чисел от 0 до 100.
3. Используя какую-либо инструментальную тестовую оболочку, разработайте тест по тестовым заданиям курса информатики, описанным в этой главе.

§12. КОМПЬЮТЕРНЫЕ ВИРУСЫ

12.1. ЧТО ТАКОЕ КОМПЬЮТЕРНЫЙ ВИРУС

Среди огромного разнообразия видов компьютерных программ существует одна их разновидность, заслуживающая особого упоминания. Главное отличие этих программ от всех остальных состоит в том, что они вредны, т.е. предназначены для нанесения ущерба пользователям ЭВМ. Это - компьютерные вирусы.

Компьютерным вирусом называется программа, обычно малая по размеру (от 200 до 5000 байт), которая самостоятельно запускается, многократно копирует свой код, присоединяя его к кодам других программ («размножается») и мешает корректной работе компьютера и/или разрушает хранимую на магнитных дисках информацию (программы и данные).

Существуют вирусы и менее «злонамеренные», вызывающие, например, переустановку даты в компьютере, музыкальные (проигрывающие какую-либо мелодию), приводящие к появлению на экране дисплея какого-либо изображения или к искажениям в отображении дисплеем информации, «осыпанию букв» и т.д.

Создание компьютерных вирусов можно квалифицировать с юридической точки зрения как преступление.

Интересны причины, заставляющие квалифицированных программистов создавать компьютерные вирусы, ведь эта работа не оплачивается и не может принести известности. Повидимому, для создателей вирусов это способ самоутверждения, способ доказать свою квалификацию и способности. Созданием компьютерных вирусов занимаются квалифицированные программисты, по тем или иным причинам не нашедшие себе места в полезной деятельности, в разработке прикладных программ, страдающие болезненным сомнением или комплексом неполноценности. Становятся создателями вирусов и те молодые программисты, которые испытывают трудности в общении с окружающими людьми, не встречают признания со стороны специалистов, которым чужды понятия морали и этики компьютерной сферы деятельности.

Существуют и такие специалисты, которые отдают свои силы и талант делу борьбы с компьютерными вирусами. В России - это известные программисты Д.Лозинский, Д.Мостовой, П.А.Данилов, Н.Безруков и др. Ими исследованы многие компьютерные вирусы, разработаны антивирусные программы, рекомендации по мерам, предотвращающим уничтожение вирусами компьютерной информации и распространение эпидемий компьютерных вирусов.

Главную опасность, по их мнению, представляют не сами по себе компьютерные вирусы, а пользователи компьютеров и компьютерных программ, не подготовленные к встрече с вирусами,

ведущие себя неквалифицированно при встрече с симптомами заражения компьютера, легко впадающие в панику, что парализует нормальную работу.

12.2. РАЗНОВИДНОСТИ КОМПЬЮТЕРНЫХ ВИРУСОВ

Рассмотрим подробнее основные особенности компьютерных вирусов, характеристики антивирусных программ и меры защиты программ и данных от компьютерных вирусов в наиболее распространенной операционной системе MS DOS.

По приближенным оценкам к 1997 г. существовало около 7000 различных вирусов. Подсчет их осложняется тем, что многие вирусы мало отличаются друг от друга, являются вариантами одного и того же вируса и, наоборот, один и тот же вирус может менять свой облик, кодировать сам себя. На самом деле основных принципиальных идей, лежащих в основе вирусов, не очень много (несколько десятков).

Среди всего разнообразия вирусов следует выделить следующие группы:

- **загрузочные (boot) вирусы** заражают программу начальной загрузки компьютера, хранящуюся в загрузочном секторе дискеты или винчестера, и запускающиеся при загрузке компьютера;

- **файловые вирусы** в простейшем случае заражают пополняемые файлы, но могут распространяться и через файлы документов (системы Word for Windows) и даже вообще не модифицировать файлы, а лишь иметь к ним какое-то отношение;

- **загрузочно-файловые вирусы** имеют признаки как загрузочных, так и файловых вирусов;

- **драйверные вирусы** заражают драйверы устройств компьютера или запускают себя путем включения в файл конфигурации дополнительной строки.

Из вирусов, функционирующих не на персональных компьютерах под операционной системой MS DOS, следует упомянуть **сетевые вирусы**, распространяющиеся в сетях, объединяющих многие десятки и сотни тысяч компьютеров.

Рассмотрим принципы функционирования загрузочных вирусов. На каждой дискете или винчестере имеются служебные сектора, используемые операционной системой для собственных нужд, в том числе сектор начальной загрузки. В нем помимо информации о дискете (число дорожек, число секторов и пр.) хранится небольшая программа начальной загрузки, о которой уже рассказывалось в настоящей главе.

Простейшие загрузочные вирусы, резидентно находясь в памяти зараженного компьютера, обнаруживают в дисковом пространстве незараженную дискету и производят следующие действия:

- выделяют некоторую область дискеты и делают ее недоступной операционной системе (помечая, например, как сбойную - bad);
- замещают программу начальной загрузки в загрузочном секторе дискеты, копируя корректную программу загрузки, а также свой код, в выделенную область дискеты;
- организуют передачу управления так, чтобы вначале выполнялся бы код вируса и лишь затем - программа начальной загрузки.

Магнитные диски компьютеров винчестерского типа обычно бывают разбиты на несколько логических разделов. Программы начальной загрузки при этом имеются в MBR (Master Boot Record - главная загрузочная запись) и в загрузочном разделе винчестера, заражение которых может происходить аналогично заражению загрузочного сектора дискеты. Однако, программа начальной загрузки в MBR использует при переходе к программе загрузки загрузочного раздела винчестера, так называемую таблицу разбиения (Partition table), содержащую информацию о положении загрузочного раздела на диске. Вирус может исказить информацию Partition table и таким образом передать управление своему коду, записанному на диск, формально не меняя загрузочной программы.

Теперь рассмотрим принципы функционирования **файловых вирусов**. Файловый вирус не обязательно является резидентным, он может, например, внедриться в код исполняемого файла. При запуске зараженного файла вирус получает управление, выполняет некоторые действия и возвращает управление коду, в который он был внедрен. Действия, которые выполняет вирус, включают поиск подходящего для заражения файла, внедрение в него так, чтобы получить управление при запуске файла, произведение некоторого эффекта, например, звукового или графического. Ес-

ли файловый вирус резидентный, то он устанавливается в памяти и получает возможность заражать файлы и проявляться независимо от первоначального зараженного файла.

Заражая файл, вирус всегда изменит его код, но далеко не всегда производит другие изменения. В частности, может не изменяться начало файла и его длина (что раньше считалось признаком заражения). Например, вирусы могут искажать информацию о файлах, хранящуюся в служебной области магнитных дисков -таблице размещения файлов (FAT - file allocation table), - и делать таким образом невозможной любую работу с файлами. Так ведут себя вирусы семейства «Dir».

Загрузочно-файловые вирусы используют принципы как загрузочных, так и файловых вирусов, и являются наиболее опасными.

12.3. АНТИВИРУСНЫЕ СРЕДСТВА

К настоящему времени накоплен значительный опыт борьбы с компьютерными вирусами, разработаны антивирусные программы, известны меры защиты программ и данных. Происходит постоянное совершенствование, развитие антивирусных средств, которые в короткий срок с момента обнаружения вируса -от недели до месяца - оказываются способными справиться с вновь появляющимися вирусами.

Создание антивирусных программ начинается с обнаружения вируса по аномалиям в работе компьютера. После этого вирус тщательно изучается, выделяется его сигнатура - последовательность байтов, которая полностью характеризует программу вируса (наиболее важные и характерные участки кода), выясняется механизм работы вируса, способы заражения. Полученная информация позволяет разработать способы обнаружения вируса в памяти компьютера и на магнитных -дисках, а также алгоритмы обезвреживания вируса (если возможно, удаления вирусного кода из файлов - «лечения»).

Известные ныне антивирусные программы можно разделить на несколько типов, перечисленных ниже.

- **Детекторы.** Их назначение - лишь обнаружить вирус. Детекторы вирусов могут сравнивать загрузочные сектора дискет с известными загрузочными секторами, формируемыми операционными системами различных версий, и таким образом обнаруживать загрузочные вирусы или выполнять сканирование файлов на магнитных дисках с целью обнаружения сигнатур известных вирусов. Такие программы в чистом виде в настоящее время редки.

- **Фаги.** Фаг - это программа, которая способна не только обнаружить, но и уничтожить вирус, т.е. удалить его код из зараженных программ и восстановить их работоспособность (если возможно). Известнейшим в России фагом является Aidstest, созданный Д.Лозинским. К январю 1997 года эта программа была способна обнаружить и обезвредить около 1600 вирусов. Ежедневно появляются новые версии этой программы, рассчитанные на обезвреживание десятков новых вирусов.

Очень мощным и эффективным антивирусным средством является фаг Doctor Web (созданный И.Даниловым). Детектор этого фага не просто сканирует файлы в поисках одной из известных вирусных сигнатур. Doctor Web реализует эвристический метод поиска вирусов, может находить и обезвреживать, так называемые, полиморфные вирусы (не имеющие определенной сигнатуры), проверять файлы, находящиеся в архивах. Для нахождения вирусов Doctor Web использует программную эмуляцию процессора, т.е. он моделирует выполнение остальных файлов с помощью программной "модели микропроцессора 1-8086 и тем самым создает среду для проявления вирусов и их размножения. Таким образом, программа Doctor Web может бороться не только с полиморфными вирусами, но и с вирусами, которые только еще могут появиться в перспективе. Специалисты рекомендуют использовать Aidstest и Doctor Web в комплексе.

- **Ревизоры.** Программа-ревизор контролирует возможные пути распространения программ-вирусов и заражения компьютеров. Программы-ревизоры относятся к самым надежным средствам защиты от вирусов и должны входить в арсенал каждого пользователя. Ревизоры являются единственным средством, позволяющим следить за целостностью и изменениями файлов и системных областей магнитных дисков. Наиболее известна в России программа-ревизор ADInf, разработанная Д.Мостовым.

- **Сторожа.** Сторож - это резидентная программа, постоянно находящаяся в памяти компьютера, контролирующая операции компьютера, связанные с изменением информации на магнит-

ных дисках, и предупреждающая пользователя о них. В состав операционной системы MS DOS, начиная с версии 6.0, входит сторож VSAFE. Однако, из-за того, что обычные программы выполняют операции, похожие на те, что делают вирусы, пользователи обычно не используют сторожа, так как постоянные предупреждения мешают работе.

• **Вакцины.** Так называются антивирусные программы, ведущие себя подобно вирусам, но не наносящие вреда. Вакцины предохраняют файлы от изменений и способны не только обнаружить факт заражения, но и в некоторых случаях «вылечить» пораженные вирусами файлы. В настоящее время антивирусные программы-вакцины широко не применяют, так как в прошлые годы некоторыми некорректно работающими вакцинами был нанесен ущерб многим пользователям.

Помимо программных средств защиты от вирусов существуют и специальные дополнительные устройства, обеспечивающие надежную защиту определенных разделов винчестера. Примером такого рода устройств является плата Sheriff (разработанная Ю.Фоминым). Несмотря на кажущееся обилие программных антивирусных средств, даже все вместе они не обеспечивают полной защиты программ и данных, не дают 100%-ной гарантии от воздействия вирусных программ. Только комплексные профилактические меры защиты обеспечивают надежную защиту от возможной потери информации. В комплекс таких мер входит:

- регулярное архивирование информации (создание резервных копий важных файлов и системных областей винчестера);
- избегание использования случайно полученных программ (старайтесь пользоваться только законными путями получения программ);
- входной контроль нового программного обеспечения, поступивших дискет;
- сегментация жесткого диска, т.е. разбиение его на логические разделы с разграничением доступа к ним;
- систематическое использование программ-ревизоров для контроля целостности информации;
- при поиске вирусов (который должен происходить регулярно!) старайтесь использовать заведомо чистую операционную систему, загруженную с дискеты. Защищайте дискеты от записи, если есть хоть малая вероятность заражения.

При неаккуратной работе с антивирусными программами можно не только переносить с ними вирусы, но и вместо лечения файлов безнадежно их испортить. Полезно иметь хотя бы общее представление о том, что могут и чего не могут компьютерные вирусы, об их жизненном цикле, о важнейших методах защиты.

Контрольные вопросы и задания

1. Что называется компьютерным вирусом?
2. Какие типы компьютерных вирусов существуют?
3. Каковы принципы функционирования загрузочных вирусов?
4. Каковы принципы функционирования файловых вирусов?
5. Охарактеризуйте известные типы антивирусных программ.
6. Перечислите меры защиты информации от компьютерных вирусов.

§ 13. КОМПЬЮТЕРНЫЕ ИГРЫ

13.1. ВИДЫ И НАЗНАЧЕНИЕ КОМПЬЮТЕРНЫХ ИГР

Игры любят все. Игра является наилучшей средой для обучения любому виду деятельности.

Характерной приметой компьютерной эры стали компьютерные игры. К ним можно относиться по-разному. С одной стороны, они могут приносить пользу как учебные средства; с другой - отнимать время (иногда очень много), отвлекать от работы. Никуда не годится играть в рабочее время, применять для этого свои дискеты, рискуя занести вирус в систему коллективного пользования и т.п. Психологи считают, что игры с изобилием сцен насилия - пусть даже выраженного в предельно условной форме - способствуют формированию у детей не самых лучших качеств. Тем

не менее, компьютерные игры широко распространены. Над их созданием трудятся высокопрофессиональные программисты, художники, мультипликаторы; это достаточно процветающая подотрасль индустрии программирования. Как и многие человеческие достижения, компьютерные игры можно использовать во благо и во зло.

Компьютерные игры ориентируются на развитие у игроков определенных знаний, навыков, способностей. Как правило, в компьютерных играх от игрока требуется

- владение средствами управления, быстрота и точность манипуляций;
- быстрая и правильная реакция на происходящие события;
- чувство времени, умение выдерживать заданные временные интервалы;
- способность следить за несколькими объектами одновременно;
- знание географии игрового поля, законов игрового мира;
- знание конкретной предметной области, которая моделируется в игре;
- умение искать закономерности;
- умение предугадывать действия противника;
- знание алгоритма и стратегии выигрыша;
- способность к быстрому и максимально полному перебору основных вариантов;
- память на текущие события;
- использование прошлого опыта, что происходило в предыдущих сеансах игры;
- способность интенсивно работать в течение всего сеанса игры.

В разных играх необходимы разные качества. Привлекательность компьютерных игр определяется следующими факторами:

- интересным сценарием;
- богатым внешним оформлением;
- кажущейся простотой;
- бесконечностью игры (недостижимостью поставленной цели);
- наличием большого числа стратегий;
- разнообразием игровых ситуаций.

В компьютерных играх можно выделить следующие категории:

- игры на мастерство;
- азартные игры;
- логические игры;
- обучающие (дидактические) игры.

Игры на мастерство основаны на управлении игровыми объектами. В азартных играх исход в большей степени зависит от случайности, везения. Логические игры содержат стратегию поведения игрока, зависящую от игровых ходов соперника или от игровой ситуации. В обучающих играх объектом управления становится ученик, а целью - отработка некоторых навыков и усвоение знаний.

По способам реализации игры можно классифицировать по признакам:

- дискретные и игры с режимом реального времени;
- антагонистические и неантагонистические;
- конечные и бесконечные;
- со случайными событиями или детерминированные;
- для одного или двух и более участников;
- игры с разным уровнем сложности.

Самые распространенные компьютерные игры - пошаговые, конечные, детерминированные для двух участников, один из которых компьютер.

По структуре в компьютерных играх можно выделить три блока и три уровня:

- блок игровой среды (правила игры);
- блок взаимодействия с играющим (интерфейс);
- блок оценки игровой ситуации (анализ);
- уровень оперативный (текущее управление клавишами);
- уровень тактический (локальные цели, усложнение игры);
- уровень стратегический (конец игры, фиксация результатов).

Общая структура компьютерных игр представлена на рис. 2.31.

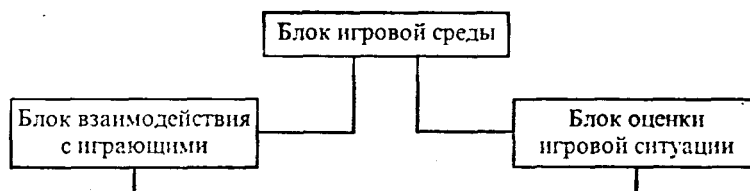


Рис. 2.31. Общая структура компьютерных игр

Блок игровой среды - это та сцена, тот трехмерный компьютерный мир, в котором есть все, что стоит, лежит, движется, появляется и исчезает в соответствии со смыслом и законами игры.

Блок взаимодействия - это все то в программе, что позволяет игроющему изменять то, что предусмотрено блоком игровой среды.

Блок оценки - это условия для игроющего и для объектов игры на игровой сцене. Это подсчет числа очков, описание или показ начальной и конечной игровой ситуации в игре.

Удастся выделить три иерархических уровня, которые позволяют правильно построить схему игры: оперативный, тактический и стратегический.

Оперативный уровень - это изменение объектов на игровой сцене посредством нажатия клавиш или управляющего устройства (мышь, джойстик). Результатом действия оперативного уровня должно быть отображение всех перемещений и изменений на экране дисплея.

Тактический уровень включает и оперативный. Действия на этом уровне ведут к достижению некоторой вполне определенной локальной цели. Изменения сложности игры, темпа, уровня происходят на этом этапе.

Стратегический уровень включает тактический и содержит несколько самостоятельных блоков: ввод на игровую схему всех объектов для определения, задания и визуализации их начальных параметров, проверка критериев окончания игры, фиксации и визуализации результатов всей игры в целом и результатов прошлых игр.

13.2. ОБЗОР КОМПЬЮТЕРНЫХ ИГР

Компьютерные игры бывают разные и каждая из них требует разных ресурсов компьютера. Ниже приведен краткий обзор компьютерных игр.

Игры на мастерство

1. Имитационные игровые виды спорта: футбол, волейбол, теннис и т.д.
2. Военные игры: морские бои, воздушные бои, звездные войны, игры с вооружениями и др.
3. Рукопашные схватки: каратэ, кунг-фу, тэквондо.
4. Профессиональные игры: авторалли, пилотирование самолета и др.
5. Приключенческие игры: путешествия, подвиги, приключения.
6. Графические игрушки: тетрис, выбивалки, «живые картинки» и т.д.
7. Учебные тренажеры: азбука, клавиатурный тренажер и др.

Азартные игры

1. Карточные игры: пасьянсы, покер, преферанс, марьяж, бридж и т.п.
2. Имитационные азартные игры: кости, рулетка, «поле чудес» и др.

Логические игры

1. Шахматные игры: шашки, шахматы, реверси и т.п.
2. Логические учебные развивающие игры: крестики - нолики, минер, лабиринты, угадан число, слово, быки и коровы, поле чудес, ним и др.

Обучающие игры

Особая роль в мире компьютерных игр принадлежит обучающим и развивающим играм. Педагогам хорошо известна важность и высокая эффективность игровых форм обучения, особенно успешно применяемых при работе с детьми дошкольного и младшего школьного возраста. Существуют, например, специальные среды для начального обучения информатике с забавными исполнителями, которые в игровой форме отрабатывают простые пользовательские навыки, навыки алгоритмизации и т.д. Так, созданная под руководством Ю.А.Первина среда «Роботландия» с ее забавными исполнителями «Машинист», «Буквояд», «Кукарачча» и др. с успехом используется во многих школах.

До недавнего времени основная масса игр требовала минимальных компьютерных ресурсов. Современная индустрия компьютерных игр нацелена на использование технологии мультимедиа и виртуальной реальности. Для них требуются высокие характеристики компьютера и его периферии.

Контрольные вопросы

1. В чем полезность компьютерных игр? Может ли компьютерная игра приносить вред?
2. На какие основные группы можно разделить компьютерные игры и в чем их различия?

2. *Ахметов К. С.* Курс молодого бойца. - М.: Компьютер-пресс.1995.
3. *Бабий А. А., Коновалова Н.И., Пак Н.И., Парашей М.Г.* Методическое пособие по работе с IBM PC- Красноярск: КГПИ. «Демос», 1991.
4. *Безруков Н. Н.* Компьютерные вирусы. - М.: Наука, 1991.
5. *Брукс Ф. П.* Как проектируются и создаются программные комплексы. /Очерки по системному программированию. - М.: Наука. 1979.
6. *Брябрин В. М.* Программное обеспечение персональных ЭВМ. - М.: Наука, 1990.
7. *Богумирский Б. С.* Руководство пользователя ПЭВМ. Части 1,2. - С.-Пб.: Печатный Двор, 1994.
8. *К.Герман О. В.* Введение в теорию экспертных систем и обработку знаний. -Минск: «Дизайн-ПРО», 1995.
9. *Глазко Т.П., Киреева Т. В., Киреев Н.В.* Работа с инструментальными средствами на IBM. - Красноярск, КГПУ, 1995.
10. *Илюшечкин В.М., Костин А.Е.* Системное программное обеспечение. - М.: Высшая школа, 1991.
11. Как работать с Microsoft Office для Windows 95. Техническое описание Microsoft Corporation. 1995.
12. *Каратыгин С., Тихонов А., Долголаптев В.* Базы данных: простейшие средства обработки информации. Т. 1,2. Серия «Компьютер для носорога». - М.: ФИА, 1995.
13. *Келжес А. М. и др.* Работа на IBM. - М.:АО «Книга и бизнес». 1992.
14. *Кирмайер М.* Мультимедиа. - С.-Пб.: ВHV-Санкт-Петербург, 1994.
15. *Климов Д. М., Руденко В.М.* Методы компьютерной алгебры в задачах механики. - М.: Наука, 1989.
16. Компьютерная вирусология. - Киев: Укр. сов. энцикл., 1991.
17. *Косневски Ч.* Занимательная математика и персональный компьютер. - М.: Просвещение, 1989.
18. *Липаев В. В.* Проектирование программных средств. - М.: Высшая школа, 1990.
19. *Маейрс Г.* Надежность программного обеспечения. - М.: Мир, 1980.
20. *Наймертаим Д.* Word 6.0 для Windows. - М.: Международные отношения, 1995.
21. Операционная система MS DOS. - М.: Радио и связь, 1992.
22. *Одинцова О. П.* Введение в Автокад. Методические указания. - Красноярск, КГПУ, 1996.
23. *Пак Н. И. Рогов В В* Компьютерная графика. - Омск, ОмГПУ, 1995.
24. Проектирование пользовательского интерфейса на персональных компьютерах. Стандарт фирмы IBM. - Вильнюс: DBS Ltd., 1992.
25. *Ратбон Э.* Windows 3.1 для «чайников». - Киев: Диалектика, 1994.
26. *Роджерс Д. Ф., Адамс Дж.А.* «Математические основы машинной графики». -М: Машиностроение, 1980.
27. *Скляров В. А.* Программное и лингвистическое обеспечение персональных ЭВМ. Системы общего назначения. - Минск: Высшая школа, 1992.
28. *Смирнов Н.Н.* Программные средства персональных ЭВМ. - Л.: Машиностроение, 1990.
29. *Смолянинова О. Г.* Основы компьютерной грамотности. - Красноярск, КГПУ, 1996.
30. *Смолянинова О. Г., Яшин А. В.* СУБД MS Access. Метод, указания. - Красноярск, КГПУ, 1997.
31. *Соловьев Г.Н., Никитин В. Д.* Операционные системы ЭВМ. - М.: Высшая школа, 1989.
32. *Туранова Л. М.* Элементы компьютерной графики. Уч. пособие. - Красноярск, КГПУ, 1996.
33. *Фигурнов В.Э.* IBM PC для пользователя. Изд.б-е. - М.: ИНФРА, 1995.
34. *Фокс Дж.* Программное обеспечение и его разработка. - М.: Мир, 1985.
35. *Фош Д., Ван Дем А.* Основы интерактивной машинной графики. - М.: Мир, 1989.
36. *Хаслер Р., Фаненистих К.* Word 6.0 для Windows. - М.: Экон, 1994.
37. *Шафрин Ю.* Основы компьютерной технологии. - М.: АБФ, 1996.
38. *Шенен П., Каспар М. и др.* Математика и САПР, книга 1. - М.: Мир, 1988.
39. *Шикин Е. В.* Начала компьютерной графики. - М.: Диалог-МИФИ, 1994.
40. *Шкаев А. В.* Настольные издательские системы: Справочник. - М.: Радио и связь, 1994.

ГЛАВА 3

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

Прохожий спросил у фермера, сооружающего небольшую деревянную постройку, что тот строит. - Пока не знаю, -ответил фермер. Если мне удастся сдать это сооружение, тогда оно - очаровательный сельский домик, а если не удастся, то это будет курятник.

Из журнала Reader's Digest

ВВЕДЕНИЕ

Данная глава посвящена важнейшему разделу информатики - программированию ЭВМ. В ней рассматриваются конкретные языки программирования, являющиеся наиболее употребимыми в настоящее время и отражающие различные тенденции в современном программировании. По каждому из языков приводятся необходимые сведения и примеры, чтобы сложилась общая картина и стало возможным самостоятельное решение относительно несложных задач по программированию. При необходимости изучить тот или иной язык «полностью» (задача эта весьма неопределенная хотя бы из-за наличия у каждого языка многих версий, особенностей трансляторов, интерфейсных оболочек и т.д.) необходимо обратиться к специальной литературе.

Наибольшее внимание уделено языку Паскаль. Он заслуженно является наиболее популярным при традиционном - процедурном - подходе к программированию, пригоден для разработки прикладных программ для самых различных предметных областей. Именно на базе Паскаля создана одна из наиболее мощных сред объектно-ориентированного программирования, что является дополнительным стимулом к его более детальному изучению.

§ 1. ИСТОРИЯ РАЗВИТИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Одной из самых революционных идей, приведших к созданию автоматических цифровых вычислительных машин, была высказанная в 20-х годах XIX века Ч.Бэббиджем мысль о предварительной записи порядка действий машины для последующей автоматической реализации вычислений - программе. И, хотя использованная Бэббиджем запись программы на перфокартах, придуманная для управления ткацкими станками французским изобретателем Жозефом Мари-Жаккаром, технически не имеет ничего общего с современными приемами хранения программ в ЭВМ, принцип здесь по- существу один. С этого момента начинается история программирования.

Аду Лавлейс, одну из немногих современников Чарльза Бэббиджа, кто сумел по достоинству оценить аналитическую машину, называют первым в мире программистом. Она теоретически разработала некоторые приемы управления последовательностью вычисления, которые используются в программировании и по сей день, описала одну из важнейших конструкций практически любого современного языка программирования - цикл.

Революционным моментом в истории языков программирования стало появление системы кодирования машинных команд с помощью специальных символов, предложенной Джоном Моучли, сотрудником Пенсильванского университета. Система кодирования, предложенная Моучли, увлекла одну из сотрудниц его компании - Грейс Мюррей Хоппер, которая посвятила всю свою жизнь компьютерам и программированию. Она вспоминает, что стала «третьим в мире программистом первого в мире большого цифрового компьютера». Г.Хоппер доказала, чего она стоит как программист. Впоследствии она писала: «Я имела то преимущество, что изучала как технику, так и математику и знала, как работает машина от начала и до конца».

При работе на компьютере «Марк-1» Г.Хоппер и ее группе пришлось столкнуться со многими проблемами и все, что ими придумано, было впервые. В частности, они придумали подпрограммы. Сейчас любой программист не задумываясь использует аппарат подпрограмм в любом

языке программирования. И еще одно фундаментальное понятие техники программирования впервые ввели Г.Хоппер и ее группа - «отладка». Однажды жарким летним днем 1945 г. неожиданно произошла остановка компьютера «Марк-1». Обнаружилась неисправность одного реле, контакты которого были заблокированы мотыльком, залетевшим неизвестно каким образом в помещение вычислительного центра. Вспоминает Г.Хоппер: «Когда к нам зашел офицер, чтобы узнать, чем мы занимаемся, мы ответили, что очищаем компьютер от насекомых (debuging)». С тех пор термин «debuging» (отладка) используется в технических процессах тестирования неисправностей в компьютере, а также в системах программирования.

На заре компьютерной эры машинный код был единственным средством общения человека с компьютером. Огромным достижением создателей языков программирования было то, что они сумели заставить сам компьютер работать переводчиком с этих языков на машинный код.

В конце 40-х годов, до прихода Г.Хоппер в фирму Джона Моучли, последний создал систему под названием «Short Code», которая являлась примитивным языком программирования высокого уровня. В ней программист записывал решаемую задачу в виде математических формул, а затем, используя специальную таблицу, переводил символ за символом, преобразовывал эти формулы в двухлитерные коды. В дальнейшем специальная программа компьютера превращала эти коды в двоичный машинный код. Система, разработанная Дж. Моучли, была по существу одним из первых примитивных интерпретаторов.

Уже в 1951 г. Хоппер создала первый в мире компилятор и ею же был введен сам этот термин. Компилятор Хоппер осуществлял функцию объединения команд и в ходе трансляции производил организацию подпрограмм, выделение памяти компьютера, преобразование команд высокого уровня (в то время псевдокодов) в машинные команды. «Подпрограммы находятся в библиотеке (компьютера), а когда вы подбираете материал из библиотеки - это называется компиляцией» - так она объясняла происхождение введенного ею термина.

В 1954 г. группа под руководством Г.Хоппер разработала систему, включающую язык программирования и компилятор, которая в дальнейшем получила название MATH-MATIC. После удачного завершения работ по созданию MATH-MATIC Г.Хоппер и ее группа принялись за разработку нового языка и компилятора, который позволил бы пользователям программировать на языке, близком к обычному английскому. Необходимость появления подобной системы Хоппер объясняла следующим образом: «Существует много различных людей, которым нужно решать разные задачи. Некоторые из них связаны с обработкой символов, другие - с обработкой слов, и им нужны языки другого типа, а не наши попытки превратить их всех в математиков». В 1958 г. появился компилятор FLOW-MATIC. В отличие от ФОР-ТРАНа - языка для научных приложений - FLOW-MATIC был первым языком для задач обработки коммерческих данных. Работы в этом направлении привели к созданию языка КОБОЛ (COBOL - Common Business Oriented Language). Одним из основных консультантов при создании этого языка была Грейс Мюррей Хоппер.

Середина 50-х годов характеризуется стремительным прогрессом в области программирования. Роль программирования в машинных командах стала уменьшаться. Начали появляться языки программирования нового типа, выступающие в роли посредника между машинами и программистами. Первым и одним из наиболее распространенных был Фортран (FORTRAN, от FORMula TRANslator - переводчик формул), разработанный группой программистов фирмы IBM в 1954 г. (первая версия).

В середине 60-х годов сотрудники математического факультета Дартмутского колледжа Томас Курц и Джон Кемени создали специализированный язык программирования, который состоял из простых слов английского языка. Новый язык называли «универсальным символическим кодом для начинающих» (Beginners All-Purpose Symbolic Instruction Code, или, сокращенно, BASIC). Годом рождения нового языка можно считать 1964 г. Сегодня универсальный язык Бейсик (имеющий множество версий) приобрел большую популярность и получил широкое распространение среди пользователей ЭВМ различных категорий во всем мире. В значительной мере этому способствовало то, что Бейсик начали использовать как встроенный язык персональных компьютеров, широкое распространение которых началось в конце 70-х годов.

В начале 60-х годов все существующие языки программирования высокого уровня можно было пересчитать по пальцам, однако впоследствии их число достигло трех тысяч. Разумеется, подавляющая часть языков не получила сколько-нибудь широкого распространения; в практической деятельности используется не более двух десятков. Разработчики ориентировали языки на

разные классы задач, в той или иной мере привязывали их к конкретным архитектурам ЭВМ, реализовывали личные вкусы и идеи. В 60-е годы были предприняты попытки преодолеть эту «разногласицу» путем создания универсального языка программирования. Первым детищем этого направления стал PL/I (Programm Language One), 1967 г. Затем на эту роль претендовал АЛГОЛ-68 (1968 г.). Предполагалось, что подобные языки будут развиваться и усовершенствоваться и вытеснят все остальные. Однако ни одна из этих попыток на сегодняшний день не увенчалась успехом (хотя PL/I в усеченных версиях использовали многие программисты). Всеохватность языка приводила к неоправданной, с точки зрения программиста, сложности конструкций, неэффективности компиляторов.

Языки программирования служат разным целям и их выбор определяется удобностью пользователя, пригодностью для данного компьютера и данной задачи. А задачи для компьютера бывают самые разнообразные: вычислительные, экономические, графические, экспертные и т.д. Такая разнотипность решаемых компьютером задач и определяет многообразие языков программирования. По всей видимости, в программировании наилучший результат достигается при индивидуальном подходе, исходящем из класса задачи, уровня и интересов программиста. Например, Бейсик широко употребляется при написании простых программ; Фортран является классическим языком программирования при решении на ЭВМ математических и инженерных задач; язык Кобол (COBOL, от Common Business Oriented Language -общий язык, ориентированный на деловые задачи; создан в 1960 г.) был задуман как основной язык для массовой обработки данных в сферах управления и бизнеса. Еще более специализированным является язык ЛОГО (от греческого logos - слово), созданный для обучения программированию школьников профессором математики и педагогики Сеймуром Пейпертом из Массачусетского технологического института. Обучаясь программированию на ЛОГО, дети задают простые команды, которые управляют игрушечной черепашкой, снабженной карандашиком. Отметим и еще один достаточно популярный специализированный язык - Пролог (Prolog -PROgramming in LOGic), разработанный как язык программирования для создания систем искусственного интеллекта.

В конце 50-х годов плодом международного сотрудничества в области программирования явился Алгол (ALGOL, от ALGOrithmic Language - алгоритмический язык). Алгол предназначен для записи алгоритмов, которые строятся в виде последовательности процедур, применяемых для решения поставленных задач. Специалисты-практики восприняли этот язык далеко неоднозначно, но, тем не менее, его влияние на развитие других языков и теорию программирования оказалось весьма значительным.

В нашей стране в те годы был создан под руководством Сергея Петровича Ершова транслятор Альфа, который представлял довольно удачную русифицированную версию Алгола. Впоследствии академик Ершов сыграл важнейшую роль в становлении в СССР школьной информатики.

Развитие идеи Алгола о структуризации разработки алгоритмов нашло наивысшее отражение при создании в начале 70-х годов языка Паскаль швейцарским ученым Никлаусом Виртом. Язык Паскаль первоначально разрабатывался как учебный, и, действительно, сейчас он является одним из основных языков обучения программированию в школах и вузах. Однако, качества его в совокупности оказались столь высоки, что им охотно пользуются и профессиональные программисты.

Не менее впечатляющей, в том числе и финансовой, удачи добился джазист Филипп Кан, француз, разработавший систему Турбо-Паскаль. Суть его идеи состояла в объединении последовательных этапов обработки программы - компиляции, редактирования связей, отладки и диагностики ошибок - в едином интерфейсе. Версии Турбо-Паскаля заполонили практически все образовательные учреждения, программистские центры и частные фирмы.

Период с конца 60-х и до начала 80-х годов характеризуется бурным ростом числа различных языков программирования, сопровождавшим, как это ни парадоксально, кризис программного обеспечения. Этот кризис особо остро переживало военное ведомство США. В январе 1975 г. Пентагон решил навести порядок в хаосе трансляторов и учредил комитет, которому было предписано разработать один универсальный язык. На конкурсной основе комитет проработал сотни проектов и, когда стало ясно, что ни один из существующих языков не может их удовлетворить, принял два проекта для окончательного рассмотрения. В мае 1979 г. был объявлен победитель - группа ученых во главе с Жаном Ихбиа. Победивший язык окрестили АДА, в честь Огасты Ады Лавлейс. Язык АДА - прямой наследник языка Паскаль. - этот язык предназначен для создания и

длительного (многолетнего) сопровождения больших программных систем, допускает возможность параллельной обработки, правления процессами в реальном времени и многое другое, чего трудно или невозможно достичь средствами более простых языков.

Большой отпечаток на современное программирование наложил язык Си (первая версия - 1972 г.), являющийся очень популярным в среде разработчиков систем программного обеспечения (включая операционные системы). Си сочетает в себе черты как языка высокого уровня, так и машинно-ориентированного языка, допуская программиста ко всем машинным ресурсам, чего не обеспечивают такие языки, как Бейсик и Паскаль.

Следует отметить, что многие языки, первоначально разработанные для больших и малых ЭВМ, в дальнейшем были приспособлены к персональным компьютерам. Хорошо вписались в «персоналки» не только Паскаль, Бейсик, Си, Лого, но и ЛИСП, ПРОЛОГ - языки искусственного интеллекта.

В течение многих лет программное обеспечение строилось на основе операционных и процедурных языков, таких как Фортран, Бейсик, Паскаль, Ада, Си. И сегодня современные версии этих и им подобных языков (Модула, Форт и др.) доминируют при разработке прикладных программных средств. Однако по мере эволюции языков программирования получили широкое распространение и другие, принципиально иные, подходы к созданию программ.

Классическое операционное и/или процедурное программирование требует от программиста детального описания того, как решать задачу, т.е. формулировки алгоритма и его специальной записи. При этом ожидаемые свойства результата обычно не указываются. Основные понятия языков этих групп - оператор и данные. При процедурном подходе операторы объединяются в группы - процедуры. Структурное программирование в целом не выходит за рамки этого направления, оно лишь дополнительно фиксирует некоторые полезные приемы технологии программирования.

Принципиально иное направление в программировании связано с методологиями (иногда говорят «парадигмами») непроведурного программирования. К ним можно отнести объектно-ориентированное и декларативное программирование. Объектно-ориентированный язык создает окружение в виде множества независимых объектов. Каждый объект ведет себя подобно отдельному компьютеру, их можно использовать для решения задач как «черные ящики», не вникая во внутренние механизмы их функционирования. Из языков объектного программирования, популярных среди профессионалов, следует назвать прежде всего Си++, для более широкого круга программистов предпочтительны среды типа Delphi и Visual Basic.

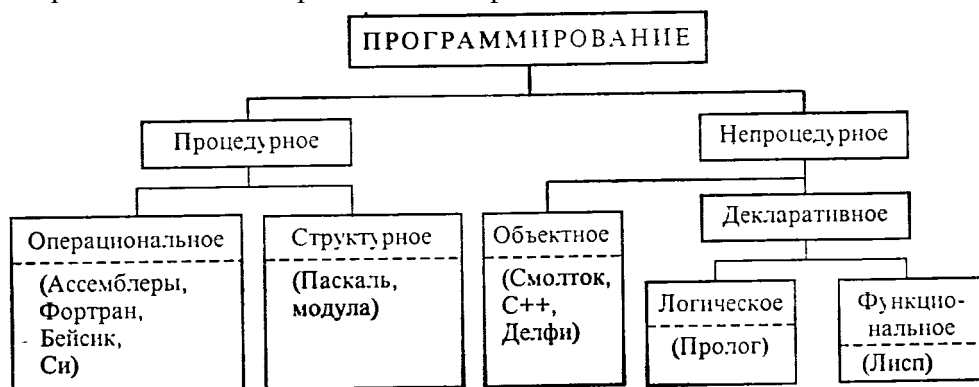


Рис. 3.1. Классификация языков программирования

При использовании декларативного языка программист указывает исходные информационные структуры, взаимосвязи между ними и то, какими свойствами должен обладать результат. При этом процедуру его получения («алгоритм») программист не строит (по крайней мере, в идеале). В этих языках отсутствует понятие «оператор» («команда»). Декларативные языки можно подразделить на два семейства - логические (типичный представитель - Пролог) и функциональные (Лисп). По всей видимости, непроведурные языки имеют большое будущее. Все сказанное выше можно отобразить схемой - крупнотруктурной классификацией языков программирования. В ней указаны основные методологии программирования; в нижнем ряду, в скобках - типичные языки соответствующих групп.

Контрольные вопросы

1. Какие события и когда стали толчком для начала работ по программированию?
2. Каковы основные вехи на пути развития программирования?
3. Какие основные направления существуют в современном программировании?

§2. ЯЗЫКИ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ

2.1. ПОНЯТИЕ О ЯЗЫКАХ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ

Языки программирования - это формальные языки специально созданные для общения человека с компьютером. Каждый язык программирования, равно как и «естественный» язык (русский, английский и т.д.), имеет алфавит, словарный запас, свои грамматику и синтаксис, а также семантику.

Алфавит - фиксированный для данного языка набор основных символов, допускаемых для составления текста программы на этом языке.

Синтаксис - система правил, определяющих допустимые конструкции языка программирования из букв алфавита.

Семантика - система правил однозначного толкования отдельных языковых конструкций, позволяющих воспроизвести процесс обработки данных.

При описании языка и его применении используют понятия языка. **Понятие** подразумевает некоторую синтаксическую конструкцию и определяемые ею свойства программных объектов или процесса обработки данных.

Взаимодействие синтаксических и семантических правил определяют те или иные понятия языка, например, операторы, идентификаторы, переменные, функции и процедуры, модули и т.д. В отличие от естественных языков правила грамматики и семантики для языков программирования, как и для всех формальных языков, должны быть явно, однозначно и четко сформулированы.

Языки программирования, имитирующие естественные языки, обладающие укрупненными командами, ориентированными на решение прикладных содержательных задач, называют языками «высокого уровня». В настоящее время насчитывается несколько сотен таких языков, а если считать и их диалекты, то это число возрастет до нескольких тысяч. Языки программирования высокого уровня существенно отличаются от машинно-ориентированных (низкого уровня) языков. Во-первых, машинная программа в конечном счете записывается с помощью лишь двух символов 0 и 1. Во-вторых, каждая ЭВМ имеет ограниченный набор машинных операций, ориентированных на структуру процессора. Как правило, этот набор состоит из сравнительно небольшого числа простейших операций, типа: переслать число в ячейку; считать число из ячейки; увеличить содержимое ячейки на +1 и т.п. Команда на машинном языке содержит очень ограниченный объем информации, поэтому она обычно определяет простейший обмен содержимого ячеек памяти, элементарные арифметические и логические операции. Команда содержит код и адреса ячеек, с содержимым которой выполняется закодированное действие.

Языки программирования высокого уровня имеют следующие достоинства:

- алфавит языка значительно шире машинного, что делает его гораздо более выразительным и существенно повышает наглядность и понятность текста;
- набор операций, допустимых для использования, не зависит от набора машинных операций, а выбирается из соображений удобства формулирования алгоритмов решения задач определенного класса;
- конструкции команд (операторов) отражают содержательные виды обработки данных и задаются в удобном для человека виде;
- используется аппарат переменных и действия с ними;
- поддерживается широкий набор типов данных.

Таким образом, языки программирования высокого уровня являются машинно-независимыми и требуют использования соответствующих программ-переводчиков (трансляторов) для представления программы на языке машины, на которой она будет исполняться.

2.2. МЕТАЯЗЫКИ ОПИСАНИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Интерпретация конструкций языка программирования должна быть абсолютно однозначной, ибо фраза на языке программирования превращается в машинный код автоматически, с помощью программы-транслятора, и любой намек на неоднозначность либо делает эту фразу непереводимой, либо приводит к ошибке. В этом отношении языки программирования значительно отличаются от естественных языков, допускающих неоднозначно интерпретируемые фразы, рассчитанные на здравый смысл и жизненный опыт человека - слушателя и исполнителя, способного додумать содержание фразы. «Додумывание» не входит в способности компьютеров, поэтому необходимы приемы описания конструкций языков программирования типа: «Оператором присваивания называется ...», причем продолжение подобной фразы на естественном языке чаще всего оказывается либо слишком громоздким, либо неоднозначным, либо и тем, и другим одновременно.

Для строгого и точного описания синтаксиса языка программирования, как правило, используют специальные **метаязыки** (языки для описания других языков). Наиболее распространенными метаязыками являются **металингвистические формулы Бэкуса - Наура** (язык БНФ) и **синтаксические диаграммы Вирта**.

Язык БНФ (называемый также языком нормальных форм) представляет компактную форму в виде некоторых формул, похожих на математические. Для каждого понятия языка существует единственная метаформула (нормальная форма). Она состоит из левой и правой частей. В левой части указывается определяемое понятие, а в правой - задается множество допустимых конструкций языка, которые объединяются в это понятие. В формуле используют специальные метасимволы в виде угловых скобок, в которых заключено определяемое понятие (в левой части формулы) или ранее определенное понятие (в ее правой части), а разделение левой и правой частей указывается метасимволом «::=», смысл которого эквивалентен словам «по определению есть».

Например, метаформулы

```
<переменная>::=A|B
<выражение>::=<переменная>|<переменная>+<переменная>|<переменная><переменная>-
<переменная>
```

означают, что в том (сугубо модельном) языке, на который эта метаформула распространяется, под термином <переменная> понимается любая из букв A или B, а под термином <выражение> - любая из следующих десяти записей: A; B; A+A; A+B; B+A; B+B; A-A; A-B; B-A; B-B. Знак | следует читать «или».

Правая часть метаформулы может содержать правило построения допустимых последовательностей. Допускаются рекурсивные определения терминов и понятий, т.е. когда в правой части формулы участвует понятие, определяемое левой частью. Например, пусть необходимо ввести понятие <двоичный код>, под которым понималась любая непустая последовательность цифр 0 и 1. Тогда простое и компактное рекурсивное определение с помощью метаформул выглядит так:

```
<двоичная цифра>::= 0|1
<двоичный код>::=<двоичная цифра>|<двоичный код> <двоичная цифра>
```

Рекурсия здесь не мешает конструктивному построению понятия <двоичный код>, так как по принятым правилам при первом обращении к рекурсивно определяемому понятию следует ограничиться нерекурсивной частью формулы, т.е. под двоичным кодом понимать двоичную цифру - 0 или 1. Но при втором обращении к метаформуле, определяющей двоичный код, мы имеем варианты (конечно, неполные) понятия <двоичный код>, и можем применить рекурсию, которая даст нам следующие варианты этого понятия: 0 1 00 01 10 11, т.е. все возможные одно- и двух-цифровые двоичные коды. Очевидно, что при следующих применениях рекурсии мы получим любой возможный двоичный код.

Для задания синтаксических конструкций произвольной длины часто используют фигурные скобки как метасимволы. Фигурные скобки означают, что конструкция может повторяться нуль или более раз. В частности, термин <двоичный код> можно определить по другому, а именно:

$\langle \text{двоичный код} \rangle ::= \langle \text{двоичная цифра} \rangle \langle \text{двоичная цифра} \rangle$

И еще, для полноты множества синтаксических конструкций, необходимо определить конструкцию $\langle \text{пусто} \rangle$:

$\langle \text{пусто} \rangle ::= \epsilon$.

В большинстве учебных пособий по программированию, технических описаний языков, метаформулы рассматриваемого языка представлены полностью.

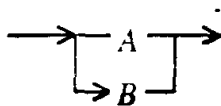
Синтаксическая диаграмма является графическим представлением значения метапеременной метаязыка. Диаграмма состоит из основных символов или понятий языка.

Каждая диаграмма имеет входящую и выходящую стрелки, означающие начало и конец синтаксической конструкции и отражающие процесс ее чтения и анализа. Из каждого элемента выходит одна или несколько стрелок, оказывающих на те элементы, которые могут следовать непосредственно за данным элементом.

Для сравнения с метаформулами приведем несколько примеров.

Синтаксическая диаграмма

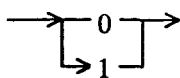
$\langle \text{переменная} \rangle ::=$



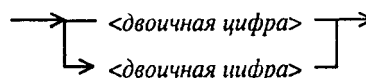
эквивалентна метаформуле $\langle \text{переменная} \rangle ::= A \mid B$.

Еще примеры:

$\langle \text{двоичная цифра} \rangle ::=$



$\langle \text{двоичный код} \rangle ::=$



Читатель может поупражняться в составлении синтаксических диаграмм для известных ему языков программирования.

Металингвистические формулы в некотором виде заложены в трансляторы; с их помощью ведется проверка конструкций, используемых программистом, на формальное соответствие какой-нибудь из конструкций, синтаксически допустимых в этом языке (синтаксический контроль).

23. ГРАММАТИКА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Описанию грамматики языка предшествует описание его алфавита. Алфавит любого языка состоит из фиксированного набора символов, однозначно трактуемых. Алфавит языков программирования, как правило, связан с литерами клавиатуры печатной машинки. Клавиатуры персональных компьютеров близки к ним по наличию литер.

Алфавиты большинства языков программирования близки друг другу и основываются на буквах латинского алфавита, арабских цифрах и общепринятых спецсимволах, таких как знаки препинания, математических операций, сравнений и обозначений. Большинство популярных языков программирования в своем алфавите содержат следующие элементы:

$\langle \text{буква} \rangle ::= \text{AaBbCcDdEeFf и т.д.}$

$\langle \text{цифра} \rangle ::= 0123456789$

$\langle \text{знак арифметической операции} \rangle ::= * / + -$

$\langle \text{разделитель} \rangle ::= \epsilon, ,, ; () [] \{ \} ' :: =$

$\langle \text{служебное слово} \rangle ::= \text{begin end if then else for next и т.д.}$

$\langle \text{спецсимвол} \rangle ::= \langle \text{знак арифметической операции} \rangle \mid \langle \text{разделитель} \rangle \mid$

$\langle \text{служебное слово} \rangle$

$\langle \text{основной символ} \rangle ::= \langle \text{буква} \rangle \mid \langle \text{цифра} \rangle \mid \langle \text{спецсимвол} \rangle$

<комментарий> ::= <любая последовательность символов>

Несмотря на значительные различия между языками программирования, ряд фундаментальных понятий в большинстве из них схожи. Приведем часть этих понятий.

Оператор - одно из ведущих понятий всех языков программирования (теоретически, за исключением чисто декларативных; но в действительности и они используют родственное понятие). Каждый оператор представляет собой законченную фразу языка и определяет однозначно трактуемый этап обработки данных. В соответствии с теорией алгоритмов выделяют основные (базисные) операторы языка: присвоения, условный и безусловный переход, пустой оператор. К производным, не основным, относят составной оператор, оператор выбора, оператор цикла и оператор присоединения.

Все операторы языка в тексте программы отделяются друг от друга явными или неявными разделителями, например:

S1;S2;...;Sn

Операторы выполняются в порядке их следования в тексте программы. Лишь с помощью операторов перехода этот естественный порядок может быть нарушен.

Большая часть операторов ведет обработку величин. **Величины** могут быть **постоянными** и **переменными**. Значения постоянных величин не изменяются в ходе выполнения программы. Величина характеризуется **типом, именем и значением**. Наиболее распространенные типы величин - числовые (целые и вещественные), символьные, логические. Тип величины определяется ее значением.

Другая важная классификация величин - простые и структурированные. Простая величина в каждый момент может иметь не более одного значения. Ей соответствует одна ячейка памяти (поскольку термин «ячейка» несколько устарел, часто говорят «машинное слово») или ее эквивалент во внешней памяти компьютера. Структурированная величина, имея одно имя, может иметь разом несколько значений. Эти значения представляют собой элементы (компоненты) величины. Самый широкоизвестный пример - массив, у которого элементы различаются по индексам (номерам). Вопрос о структурировании величин - входных, выходных и промежуточных - для успеха решения прикладной задачи не менее важен, чем вопрос о правильном написании последовательности операторов.

Важнейшие характеристики структурированной величины таковы: **упорядоченность** (да или нет), **однородность** (да или нет), **способ доступа** к элементам, **фиксированность числа элементов** (да или нет). Так, массив является упорядоченной однородной структурой с прямым доступом к элементам и фиксированным их количеством.

Всем программным объектам в языках даются индивидуальные **имена**. Имя программного объекта называют **идентификатором** (от слова «идентифицировать»). Чаще всего идентификатором является любая конечная последовательность букв и цифр, начинающаяся с буквы:

<идентификатор> ::= <буква> | <идентификатор> | <буква>
<идентификатор> <цифра>

Как правило, в большинстве языков программирования в качестве идентификатора запрещается использовать служебные слова языка.

Многим слово «идентификатор» не нравится, и в настоящее время чаще употребляют слово «имя», поскольку

<имя> ::= <идентификатор>.

Программисты выбирают имена по своему усмотрению. Принципы выбора и назначения имен программным объектам естественны. Следует избегать мало выразительных обозначений, не гоняться за краткими именами. Имена должны быть понятны, наглядны, отражать суть обозначаемого объекта. Например,

Summa, Time, i, j, integral, init и т. д.

Некоторым идентификаторам заранее предписан определенный смысл и их называют стандартными, например, Sin - это имя известной математической функции.

Описания или объявления программных объектов связаны с правилами обработки данных. Данные бывают разные и необходимо для каждого из них определить его свойства. Например, если в качестве данных выступает массив, то необходимо задать его размерность, границы индексов, тип элементов массива. Описательная часть языка программирования является необходимой как для системных программистов - разработчиков трансляторов, которые должны, в частности, проводить синтаксическую и семантическую диагностику программ, - так и для «прикладного» программиста, которому объявления программных объектов часто облегчают процесс разработки и отладки программ.

В некоторых языках стандартные описания простых числовых и символьных данных опускают (описания по умолчанию), или в них задаются правила описания по имени объекта. Например, в Фортране переменные, имена которых начинаются с букв I, J, K, L, M, N, могут принимать целые значения (при отсутствии явного описания типа, которое возможно), т.е. определены как числовые данные целого типа. В Бейсике-MSX данные строкового типа присваиваются переменным, имена которых заканчиваются специальным символом \$: A\$, S1\$.

Особый интерес представляют в языках программирования описания нестандартных структур данных, таких как запись, файл, объект, список, дерево и т.п.

Приведем список наиболее употребительных обозначений типов данных, используемых в описаниях:

<i>Целый</i>	- <i>Integer</i>
<i>Вещественный</i>	- <i>Real</i>
<i>Логический</i>	- <i>Boolean</i>
<i>Символьный</i>	- <i>Char</i>
<i>Строковый</i>	- <i>String</i>
<i>Массив</i>	- <i>Array</i>
<i>Множество</i>	- <i>Set</i>
<i>Файл</i>	- <i>File</i>
<i>Запись</i>	- <i>Record</i>
<i>Объект</i>	- <i>Object</i>

Переменные играют важнейшую роль в системах программирования. Понятие «переменная» в языках программирования отличается от общепринятого в математике. Переменная - это программный объект, способный принимать некоторое значение с помощью оператора присваивания. В ходе выполнения программы значения переменной могут неоднократно изменяться. Каждая переменная после ее описания отождествляется с некоторой ячейкой памяти, содержимое которой является ее значением. Синтаксис переменной, точнее, ее идентификатора, как правило, имеет вид:

<имя переменной> ::=
_____><буква>_____>
_____><буква>_____>
_____><цифра>_____>
_____><спецсимвол>

Семантический смысл переменной заключается в хранении некоторого значения, соответствующего ее типу (например, переменная целого типа может принимать значение произвольного целого числа), а также в выполнении с ней операций пересылки в нее и извлечения из нее этого значения.

Функция - это программный объект, задающий вычислительную процедуру определения значения, зависящего от некоторых аргументов. Вводится в языки программирования для задания программистом необходимых ему функциональных зависимостей. В каждом языке высокого уровня имеется в наличии библиотека стандартных функций: арифметических, логических, сим-

вольных, файловых и т.п. Функции -стандартные и задаваемые программистом - используются в программе в выражениях.

Выражения строятся из величин - постоянных и переменных, функций, скобок, знаков операций и т.д. Выражение имеет определенный тип, определяемый типом принимаемых в итоге его вычисления значений. Возможны выражения арифметические, принимающие числовые значения, логические, символьные, строковые и т.д. Выражение $5+7$ является, несомненно, арифметическим, выражение $A + B$ может иметь самый разный смысл - в зависимости от того, что стоит за идентификаторами A и B .

Процедура - это программный объект, представляющий некоторый самостоятельный этап обработки данных. По сути, процедуры явились преемниками подпрограмм, которые были введены для облегчения разработки программ еще на самых ранних стадиях формирования алгоритмических языков. Процедура имеет входные и выходные параметры, называемые формальными. При использовании процедуры формальные параметры заменяются на фактические.

Модуль (Unit) - это специальная программная единица, предназначенная для создания библиотек и разделения больших программ на логически связанные блоки.

По сути, модуль - это набор констант, типов данных, переменных, процедур и функций. В состав модуля входят разделы: заголовок, интерфейс, реализация, инициализация.

Заголовок необходим для ссылок на модуль.

Интерфейс содержит объявления, включая процедуры и функции.

Раздел «*реализация*» содержит тела процедур и функций, перечисленных в интерфейсной части.

Раздел «*инициализация*» содержит операторы, необходимые для инициализации модуля.

Каждый модуль компилируется отдельно, и каждый элемент модуля можно использовать в программе без дополнительного объявления.

Контрольные вопросы и задания

1. Какие преимущества имеют языки программирования высокого уровня по сравнению с машинно-ориентированными языками?
2. Каковы основные составляющие языка программирования высокого уровня?
3. В чем различия понятий языков программирования от аналогичных понятий математического «языка»?
4. С какой целью используются и что представляют собой металингвистические формулы Бэкуса-Наура?
5. Что представляет собой синтаксическая диаграмма Вирта?
6. В чем различие между постоянными и переменными величинами? Чем характеризуется величина?
7. В чем принципиальная разница между величинами простыми и структурированными?
8. Для чего служит описание величин в программах?
9. В чем состоит назначение функций? процедур? модулей?

§3. ПАСКАЛЬ КАК ЯЗЫК СТРУКТУРНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

3.1. ВВЕДЕНИЕ

Язык Паскаль, начиная с момента своего создания Н.Виртом в 1971 г., играет особую роль и в практическом программировании, и в его изучении. С непревзойденной четкостью в нем реализованы принципы структурного программирования, которые мы рассматривали в п. 1.8. Паскаль стал первым языком, с которым знакомится большинство будущих программистов в мире.

Трансляторы для программ, написанных на Паскале, разработаны для различных компьютеров и в настоящее время имеют множество разновидностей. Они являются компиляторами, обрабатывающими разработанные программистами тексты программ.

Существует много версий языка Паскаль. Различия между ними порой весьма велики. Так, базовая версия Вирта имеет многократно меньшие возможности чем, скажем, версия Турбо-

Паскаль 7.0 (первая, фактически - язык для обучения будущих программистов, а вторая - орудие профессиональных разработчиков прикладного программного обеспечения). Тем не менее, это версии одного языка, что, в частности, подтверждается их совместимостью «сверху вниз», т.е. любая программа, соответствующая «младшей» версии, соответствует и «старшей» (за исключением малозначащих синтаксических оговорок). Приведенные ниже тексты программ и примеры соответствуют (если нет специальных оговорок) версиям не ниже Турбо-Паскаль 3.0.

Любая Паскаль-программа является текстовым файлом с собственным именем и с расширением .pas. Рассмотрим в качестве примера текст программы 1 решения квадратного уравнения. Паскаль-программа имеет вид последовательности символов латинских и русских букв, арабских цифр, знаков операции, скобок, знаков препинания и некоторых дополнительных символов. В ней можно выделить описания данных и операторы, описывающие действия, которые надо выполнить машине над этими данными.

Программа 1

program KvadUravn;	(заголовок программы)
var	(список переменных)
a,b,c: real;	(коэффициенты уравнения)
d,x1,x2: real;	(вспомогательные переменные)
begin	(начало программы)
writein;	(пропуск строки на экране)
writein('введи a,b,c'); read(a,b,c);	(ввод данных)
d:=b*b-4*a*c;	(дискриминант)
if d<0 then	(если d<0, то)
write('корней нет')	(печатать)
else	(иначе)
begin	(начало серии команд)
x1:=(-b+sqrt(d))/(2*a);	
x2:=(-b-sqrt(d))/(2*a);	(вычисляем корни)
write('x1=',x1,' x2=',x2)	(печать корней)
end	(конец серии)
end.	(конец программы)

Схематически программа представляется в виде последовательности восьми разделов:

- 1) заголовок программы;
- 2) описание внешних модулей, процедур и функций;
- 3) описание меток;
- 4) описание констант;
- 5) описание типов переменных;
- 6) описание переменных;
- 7) описание функций и процедур;
- 8) раздел операторов.

Не в каждой программе обязательно присутствуют все восемь разделов, в простейшей программе, например, могут быть только 5-й и 8-й разделы.

Каждый раздел начинается со служебного слова, назначение которого зафиксировано в Паскале так, что его нельзя употреблять для других целей (список и перевод служебных слов дан в Приложении).

Рассмотрим пример программы 2, вычисляющей длину окружности и площадь круга по данному радиусу.

Программа 2

```

program circle;
const
    pi=3.14159;
var
    r,s,l : real;

```

```

begin
    writeln (введите радиус'); readln(r);
    s:=pi*r*r; l:=2*pi*r;
    writeln('площадь круга=',S:8:4); writeln('длина окружности=',
    l:8:4)
end.

```

В этой программе можно выделить четыре раздела. Описание заголовка начинается со служебного слова `program`, описание констант - `const`, описание переменных - `var`, раздел операторов начинается с `begin`. Программа заканчивается служебным словом `end`, после которого ставится точка. Описания величин и операторы друг от друга отделяются знаком «точка с запятой».

Для обозначения величин используются имена. Они состоят из латинских букв и цифр, причем первым символом должна быть буква. В примере использованы имена величин - `pi`, `r`, `s` и `l`.

Имя программы (в примере - `circle`) выбирается автором и составляется по такому же правилу.

Постоянные величины (константы) чаще всего бывают числовыми или символьными (но могут быть и других типов, о которых речь пойдет ниже). Значения символьных констант заключаются в апострофы.

Постоянные величины описываются в разделе констант по схеме:

```
const <имя> = <константа>
```

В разделе констант может быть описано несколько постоянных величин.

Например:

```
const
    pi=3.14159; k=-15; s='площадь';
```

Данные, обрабатываемые программой, могут быть разных типов (числа, символы, строки, массивы и т.д.). Тип определяет область допустимых значений, а также операции и функции, применяемые к величинам этого типа. В Паскале имеется несколько встроенных простых типов со стандартными именами.

Группа типов, значения каждого из которых можно перечислить в некотором списке - **скалярные типы**. Для них определена порядковая функция `ord(x)` - номер значения `x` в списке (для целочисленного `x` `ord(x)=x`); функции `pred(x)` - значение в списке, предшествующее `x`, и `succ(x)` - значение в списке, следующее за `x`.

Упорядоченный тип - это тип, значения которого упорядочены в обычном смысле. К данным такого типа применимы операции отношения `<`, `>`, `<=` (меньше или равно), `>=` (больше или равно), `=`, `<>` (не равно). Для логического типа выполняется неравенство:

```
false < true.
```

Переменные описываются в разделе описания переменных по схеме:

```
var <список имен переменных>: <тип>
```

Имена в списке разделяются запятой. В этом разделе может быть описано несколько переменных разного типа, например:

```
var a,b,c:real; k,l:integer; p:boolean;
```

Над **целыми величинами** (тип `integer`) определены арифметические операции : `*` (умножение), `div` (деление нацело), `mod` (вычисление остатка от деления), `+`, `-` (сложение и вычитание); операции перечислены в порядке старшинства.

Например: $25 \div 4 = 6$; $25 \bmod 4 = 1$. Целый результат дают некоторые стандартные функции (аргумент функции заключается в круглые скобки):

<code>abs(x)</code>	- абсолютная величина целого x ;
<code>sqr(x)</code>	- квадрат значения x ;
<code>trunc(x)</code>	- целая часть вещественной величины x ;
<code>round(x)</code>	- целое число, полученное из вещественного x по правилу округления;
<code>random(x)</code>	- случайное целое число из интервала от 0 до x .

Например: `trunc(4.7)=4` ; `round(4.7)=5` ; `sqr(3)=9` . Для данных типа `byte` определены те же операции и функции, что и для данных типа `integer`.

Над **вещественными величинами** определены операции: $*$, $+$, $-$, $/$, а также стандартные функции при вещественном или целом аргументе: `abs(x)`, `sqr(x)`, `sin(x)`, `cos(x)`, `arctan(x)`, `ln(x)`, `exp(x)`, `sqrt(x)` - квадратный корень из x , `int(x)` - целая часть из x , `random` - случайное число из интервала от 0 до 1. Указанные операции и функции дают вещественный результат.

Над **логическими величинами** определены операции: `not` - отрицание, `and` - конъюнкция, `or` - дизъюнкция. Логическая функция `odd(x)` принимает значение `true`, если целочисленное x является нечетным и `false` , если четным.

Множество всех символов образуют **символьные величины** (тип `char`), которые являются упорядоченными, причем `'A' < 'B' < 'C' < ... < 'Z'`, `'a' < 'b' < ... < 'z'`, `'0' < '1' < ... < '9'`.

Выражения - это конструкции, задающие правила вычисления значений переменных. В общем случае выражения строятся из переменных, констант, функций с помощью операций и скобок.

Эта роль выражений отражена в основном операторе языка - **операторе присваивания**. Он имеет следующий вид:

`<имя переменной> := <выражение>`

Тип переменной и тип выражения должны быть согласованы (величины принадлежат одному и тому же типу). Есть исключение: имя переменной может относиться к типу `real`, а значение выражения - к типу `integer`.

Примеры.

`l:=2*pi*r; p:=(a+b+c)/2; z:=sqrt(sqr(x)+sqr(y))`

В Паскале можно вводить с клавиатуры числовые и символьные данные. Имеются две встроенные процедуры (подпрограммы) ввода:

1) `read(<список переменных>)`; 2) `readln(<список переменных>)`.

При выполнении процедуры `read(x1,x2,...,xN)` программа прерывается и компьютер ждет ввода с клавиатуры N значений переменных из списка $x1, x2, ..., xN$. Эти значения - константы соответствующих типов - должны при вводе разделяться пробелами. Набор данных завершается клавишей ввода.

Процедура `readln` отличается от `read` только тем, что при завершении ввода курсор перемещается в начало следующей строки.

Пример.

```
var a,b:real; c:char; d:integer;
...
read(a,c,d,b);
...
```

Допустимый ввод: 83.14 k 200-7.15

Программа на Паскале может выводить на экран или на принтер значения числовых или символьных выражений. Имеются две процедуры вывода на экран:

1) `write(<список выражений>)`; 2) `writeln(<список выражений>)`.

Процедура `write(x1,x2,...,xN)` печатает на экране значения выражений из списка $x1, x2, ..., xN$. Процедура `writeln` отличается от `write` тем, что переводит курсор в начало следующей строки. Для вывода на принтер используются те же процедуры с добавлением служебного слова `lst` перед списком выражений.

Пример: `write(lst,'нет решений');`

На бумаге будет напечатан текст «нет решений».

Для управления печатью используются **форматы данных**. Пусть x - переменная типа `real`. Если не использовать форматы, то значение x будет выводиться в «плавающей» форме (типа 1.654887892E-04). Форматы позволяют напечатать вещественное число в естественной форме. Пусть m, n - целые числа. Процедура `write(x:m:n)` выводит на экран значение переменной x в виде десятичной дроби, причем m определяет общее число выводимых символов, включая цифры, точку и знак числа, n - количество цифр после точки. Если количество выводимых символов меньше m , то перед числом добавляются пробелы.

Пусть, например, $x = 387.26$. Следующая таблица демонстрирует влияние форматов на вывод значения x :

оператор	строка вывода
<code>writeln('*',x)</code>	<code>* 3.8726000000E+02</code>
<code>writeln('*',x:8:3)</code>	<code>* 387.260</code>
<code>writeln('*',x:8:1)</code>	<code>* 387.3</code>

Один формат - ширину поля вывода - можно использовать и для вывода значений выражений типов `integer`, `boolean`, `char`.

Контрольные вопросы

1. Какова последовательность разделов в программе на Паскале?
2. Какие типы данных называются скалярными? упорядоченными?
3. Какие действия допустимы над величинами целого типа? вещественного типа?
4. Как в Паскале осуществляется ввод и вывод данных?

3.2. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА

Паскаль - язык структурного программирования. Это означает, что программист должен выражать свои мысли очень дисциплинированно, с использованием малого числа четко оговоренных конструкций, используя как чередование их, так и вложения друг в друга. Не рекомендуется (хотя и возможно) использовать оператор перехода `goto`.

Реализация последовательности действий (т.е. структуры следования) выполняется с помощью составного оператора:

```
begin <последовательность операторов> end
```

Раздел операторов в программе всегда является составным оператором. Служебные слова `begin` и `end` часто называют операторными скобками.

Для реализации **развилки** в Паскале предусмотрены два оператора: условный оператор и оператор варианта (выбора). Они предназначены для выделения из составляющих их операторов одного, который и выполняется.

Структура и действие **условного оператора** таковы:

```
If <логическое выражение>  
then <оператор 1>  
else <оператор 2>
```

Условный оператор может быть неполным, т.е. не содержать часть «else <оператор 2>». В этом случае, если значение логического выражения равно `false`, условный оператор не вызывает никаких действий.

Пример: составим программу, которая определяет длину общей части двух отрезков числовой оси, заданных координатами своих концов соответственно a, b и c, d ($a < b, c < d$). Если отрезки имеют общую часть, то левая координата общей части отрезков m равна максимальному из чисел a и c , а правая n - минимальному из чисел b и d .

Программа 3

```
program cross;  
var a,b,c,d,m,n,l:real;  
begin  
  writeln('введите координаты концов отрезков') ;  
  read(a,b,c,d) ;
```



```
writeln;
if a<c then m:=c else m:=a;
if b<d then n:=b else n:=d;
if m<n then l:=n-m else l:=0;
writeln("длина общей части отрезков=' ,1:6:2)
end.
```

Оператор варианта имеет следующую форму:

```
case <выражение> of
    <список констант 1> : <оператор 1>;
    <список констант 2> : <оператор 2>;
    .....
    <список констант N> : <оператор N>
end.
```

Выражение, стоящее между служебными словами case и of, должно иметь значение ординального типа. Любой список констант может состоять из одной константы.

Оператор варианта вычисляет значение выражения, записанного после case. Если его значение совпадает с одной из констант в некотором списке, то выполняется оператор, стоящий после этого списка. Если значение выражения не совпало ни с одной константой во всех вариантах, то оператор варианта ничего не делает.

В качестве примера приведем программу, которая в зависимости от номера месяца выдает сообщение о времени года.

Программа 4

```
program seasons;
var k: integers
begin
    writeln('введите номер месяца') ;
    readln(k);
    case k of
        1,2,12: writeln('зима');
        3, 4,5: writeln('весна') ;
        6, 7,8: writeln('лето');
        9,10,11: writeln('осень')
    end
end.
```

Для реализации циклов в Паскале имеются три оператора. Если число повторений известно заранее, то удобно воспользоваться оператором цикла с параметром. В других случаях следует использовать операторы цикла с предусловием (цикл «пока») или с постусловием (цикл «до»).

Цикл с предусловием является наиболее мощным в Паскале. Другие операторы цикла можно выразить через него. Его форма такова:

while - <логическое выражение> **do** <оператор>

Действие: вычисляется значение логического выражения. Если оно равно true, то выполняется оператор, после чего снова вычисляется значение логического выражения, в противном случае действие заканчивается.

В качестве примера использования такого цикла решим следующую задачу. На склад привозят однородный груз на машинах различной грузоподъемности. На компьютер, управляющий работой склада, поступает информация о весе груза очередной машины. Составить программу подсчета количества машин, прибывших на склад до его заполнения, если вместимость склада не более 100 тонн.

Введем обозначения: sum - сумма веса груза, хранящегося в этот момент на складе; num - число разгруженных машин; w - масса груза очередной машины. Вначале величины sum и num равны нулю. Цикл разгрузки продолжается, пока выполняется неравенство sum < 100.

Программа 5

```
program store;
  var sum,w:real;
  num:integer;
begin
  num:=0;sum:=0;
  while sum<100 do
  begin
    writeln('введите вес груза машины');
    readln(w);
    sum:=sum+w;
    if sum<=100 then num:=num+1
    else writeln('груз уже не поместится')
    end;
    writeln('число разгруженных машин =', num;3)
  end.
end.
```

Оператор **цикла с постусловием** имеет форму:

```
repeat <последовательность операторов>
until <логическое выражение>
```

Действие: выполняется последовательность операторов. Далее вычисляется значение логического выражения. Если оно равно true, то действие заканчивается, в противном случае снова выполняется последовательность операторов и т.д.

Решим предыдущую задачу, применяя цикл с постусловием. Цикл разгрузки заканчивается, если выполняется условие: $sum > 100$.

Программа 6

```
program store2;
  var sum,w:real; man:integer;
begin
  num:=0; sum:=0;
  repeat
    writeln('введите вес груза машины');
    readln(w); sum:=sum+w;
    if sum<=100 then num:=num+1 .
    else writeln('груз уже не поместится')
  until sum>=100;
  writeln('количество разгруженных машин =', num;3)
end.
```

Оператор **цикла с параметром** предусматривает повторное выполнение некоторого оператора с одновременным изменением по правилу арифметической прогрессии значения управляющей переменной (параметра) этого цикла. Оператор цикла с параметром имеет две формы.

Форма 1:

```
for <параметр>:= <выражение 1> to <выражение 2> do <оператор>
```

Параметр, выражение 1, выражение 2 должны быть одного ординального типа;

Параметр в этом цикле возрастает. Действие эквивалентно действию следующего составного оператора:

```
begin
  <параметр>:=<выражение 1>;
  while <параметр> <= <выражение 2> do
  begin
    <оператор>;
    <параметр>:=succ(<параметр>)
  end
end.
```

Если в этом описании отношение $\leq$ заменить на $\geq$, а функцию succ на pred, то параметр в цикле будет убывать, в этом случае цикл с параметром принимает форму 2.

Форма 2:

for <параметр>:=<выражение 1> downto <выражение 2> do <оператор>

Пример: составим программу, по которой будет напечатана таблица перевода километров в мили (1 миля = 1,603 км). Параметром цикла можно считать целую величину k - количество километров. Пусть эта величина изменяется от 1 до 10 (с шагом 1, разумеется).

Программа 7

```
program mili;  
    const a=1.603; b='км'; c='мили';  
    var k:integer; m:real;  
begin  
    writeln(b:5,c:7); writeln;  
    for k:=1 to 10 do  
        begin  
            m:=k/a; writeln(k:5,m:6:3)  
        end  
end.
```

Запишем в этой программе цикл с параметром в форме 2:

```
for k:=10 downto 1 do  
begin  
    m:=k/a; writeln(k:5,m:6:3)  
end.
```

Тогда значения k в таблице будут убывать от 10 до 1 с шагом 1.

Контрольные вопросы

1. Как в Паскале реализуется развилка?
2. В чем различие в назначениях условного оператора и оператора варианта? Как реализуется оператор варианта?
3. Какого типа циклы реализуются в языке Паскаль? каким образом?

3.3. СТРУКТУРЫ ДАННЫХ

Мы уже познакомились с простыми типами real, integer, boolean, byte, char.

В Паскале программист по своему желанию может определить новый тип путем перечисления его элементов - перечисляемый тип, который относится к *простым ординальным типам*.

Описание перечисляемого типа выполняется в разделе типов по схеме:

type <имя типа> = <список имен>

Примеры:

```
type operator = (plus,minus,multi, divide);  
color = (white,red,blue,yelow,purple,green);
```

В списке должно быть не более 256 имен.

Поскольку перечисляемый тип относится к ординальным, то к его элементам можно применять функции ord(x), pred(x), succ(x) и операции отношения. Отметим, что данные этого типа не подлежат вводу и выводу с помощью функций ввода/вывода и могут использоваться внутри программы для повышения ее читабельности и понятности.

Для иллюстрации работы с перечисляемыми типами приведем программу 8, переводящую английские названия дней недели на русский язык.

Программа 8

```
program week;
  type days=(mon,tue,wed,thu,fri,sat,sun);
  var d:days;
begin
  for d:=mon to sun do
    case d of
mon: writeln("понедельник");
tue writeln("вторник");
wed writeln("среда");
thu writeln("четверг");
fri writeln("пятница");
sat writein("суббота") ;
    sun writeln("воскресенье")
    end
end.
end.
```

Интервальный тип - это подмножество другого уже определенного ординального типа, называемого *базовым*. Интервал можно задать в разделе типов указанием наименьшего и наибольшего значений, входящих в него и разделяющихся двумя последовательными точками, например:

```
type days=(mon,tue,wed,thu,fri,sat,sun);
workday s=mon.. fri;
index=1..30;
letter='a'..'z';
```

Можно задать интервал и в разделе переменных:

```
vara:1..100;b:-25..25;
```

Операции и функции - те же, что и для базового типа. Использование интервальных типов в программе позволяет экономить память и проводить во время выполнения программы контроль присваиваний.

Пример: если k - номер месяца в году, то вместо описания

```
var k:integer;
```

можно написать

```
var k:1..12;
```

Интервальный тип тоже относится к простым ординальным типам.

СОСТАВНЫЕ СТРУКТУРЫ. Данные, с которыми имеет дело ЭВМ, являются абстракциями некоторых реальных или существующих в воображении людей объектов. Мы уже познакомились с некоторыми типами данных: стандартными, перечислимыми, интервалами. Этими типами можно было обойтись при решении простых задач. Однако естественно и часто очень удобно группировать однотипные данные в последовательности - массивы, строки символов, объединять разнотипные данные об одном и том же объекте в виде записей. Значительные удобства представляются пользователю в Паскале при организации однотипных величин в виде множества с соответствующим набором операций: объединения, пересечения и т.д. Наконец, последовательность однотипных величин переменной длины можно представить в Паскале в виде файла данных и хранить на внешних носителях, используя его в разных программах.

Таким образом, подобно составному оператору, содержащему несколько других операторов, составные типы образуются из других типов, при этом существенную роль играет метод об-

разования или структура составного типа.

Часто используемый составной тип - массив. *Массив* - это последовательность, состоящая из фиксированного числа однотипных элементов. Все элементы массива имеют общее имя (имя массива) и различаются индексами. Индексы можно вычислять, их тип должен быть ординальным. При описании массивов используются служебные слова `array` и `of`. В описании массива указывается тип его элементов и типы их индексов.

Схема описания такова:

```
type <имя типа> = array [<список типов индексов>] of <тип элементов>
```

Тип элементов - произвольный, он может быть составным. Число типов индексов называется размерностью массива. После описания типа массива конкретные массивы можно задать в разделе описания переменных.

Например:

```
type vector = array [1.. 10] of real;  
table = array ['A'..'Z',1..5] of integer;  
var a,b : vector;  
c: table;
```

Обращение к элементу массива осуществляется с помощью задания имени переменной, за которым следует заключенный в квадратные скобки список индексов элемента.

Например:

```
a[7]:=3.1; b[k*k+1]:=0; c['M',3]:=-14;
```

Если массивы имеют одно и то же описание, то во многих версиях Паскаля допустимо их копирование, например `b:=a`;

Описание массива можно совместить с описанием соответствующих переменных:

```
var a,b : array [1.. 10] of real;  
d : array [byte] of char;
```

В Турбо-Паскале разрешена инициализация начальных значений составных переменных с помощью, так называемых, типизированных констант. Типизированные константы используются как переменные того же типа. Их применение экономит память, однако они не могут быть использованы для определения других переменных того же типа.

Схема описания констант массива:

```
const <имя массива>: <тип массива> = (<список значений элементов>)
```

Тип массива может быть описан ранее:

```
type digits = array [1 ..5] of char;  
const a : digits =('0';'2';'4';'6';'8');
```

Пример: используя массив, составим программу, которая напечатает на экране 20 чисел Фибоначчи.

Последовательность Фибоначчи определяется равенствами

```
a[1]=a[2]=1; a[k]=a[k-1]+a[k-2] при k>2.
```

Использование массива позволяет создать эффективную программу. Для вывода каждого члена последовательности отведем на экране 5 позиций.

Программа 9

```
program fibon; '''
    const n=20;
    var a: array[1..n] of integer;
    k: integer;
begin
    a[1]:=1;a[2]:=1;
    for k:=3 to n do a[k]:=a[k-1]+a[k-2];
    for k:=1 to n do write(a[k]:5);
    writeln
end.
```

Рассмотрим часто встречающуюся задачу упорядочения членов числовой последовательности по какому-либо признаку.

Пример: упорядочить члены числовой последовательности по возрастанию.

Используем метод упорядочения, носящий имя «пузырек». Будем просматривать пары соседних элементов последовательно справа налево и переставлять элементы в паре, если они стоят неправильно:

$5,3,2,4,1 \rightarrow 5,3,2,1,4 \rightarrow 5,3,1,2,4 \rightarrow 5,1,3,2,4 \rightarrow 1,5,3,2,4$

В начале просмотра присвоим некоторой логической переменной значение true:

p:=true; если при просмотре пар была хотя бы одна перестановка, изменим значение логической переменной на противоположное: p:=false; это означает, что последовательность еще не была упорядочена и просмотр пар надо повторить. Цикл просмотров заканчивается, если после очередного просмотра выполняется условие: p=true. Последовательность зададим в программе как константмассив из 10 элементов - целых чисел.

Программа 10

```
program bubble; '
const a:array[1..10] of integer=(19,8,17,6,15,4,13,2,11,0);
    var b,i:integer; p :boolean;
begin clrscr;
    for i:=1 to 10 do write(a[i]:3);writeln;writeln;
        repeat p:=true;
            for i:=10 downto 2 do
                if a[i]<a[i-1]
                then begin
                    b:=a[i];a[i]:=a[i-1]; a[i-1]:=b;p:=false
                end
            until p=true;
            for i:=1 to 10 do write(a[i]:3);
        writeln
    end.
```

Обработка элементов двумерных массивов (матриц) обычно выполняется с помощью двойного цикла. Один цикл управляет номером строки, другой - номером столбца.

При решении задач на ЭВМ часто возникает необходимость в использовании последовательностей символов. Такую последовательность можно описать как массив символов, однако в Паскале для таких целей имеется специальный тип - string[n] - *строка* из n символов, где n ≤ 255. Способы описания переменных - строк - аналогичны описанию массивов.

1. Строковый тип определяется в разделе описания типов, переменные этого типа - в разделе описания переменных:

```
type word : string[20];  
var a,b,c : word;
```

2. Можно совместить описание строкового типа и соответствующих переменных в разделе описания переменных:

```
var a,b,c : string[20];  
d : string[30];
```

3. Можно определить строковую переменную и ее начальное значение как констант-строку:
`const l:string[1 1]='информатика';`

Символы, составляющие строку, занумерованы слева направо; к ним можно обращаться с помощью индексов, как к элементам одномерного массива.

Для переменных одного строкового типа определен лексикографический порядок, являющийся следствием упорядоченности символьного типа:

'fife' < 'tree' (так как 'f' < 't'); '4' > '237' (так как '4' > '2').

Кроме логических операций <, >, =, для величин строкового типа определена некоммутативная операция соединения, обозначаемая знаком плюс:

`a:='кол'+ 'о'+ 'кол';` (в результате `a='колокол'`).

Для строковых величин определены следующие четыре стандартные функции.

1. Функция соединения - `concat(s1,s2,...,sk)`. Значение функции - результат соединения строк `s1,s2,...,sk`, если он содержит не более 255 символов.

2. Функция выделения - `copy(s,i,k)`. Из строки `s` выделяется `k` символов, начиная с `i`-того символа:

`a:=copy('крокодил',4,3);` (в результате `a='код'`).

3. Функция определения длины строки - `length(s)`. Вычисляется количество символов, составляющих текущее значение строки `s`:

`b:=length('каникулы');` (`b=8`).

4. Функция определения позиции - `pos(s,t)`. Вычисляется номер позиции, начиная с которого строка `s` входит первый раз в строку `t`; результат равен 0, если строка `s` не входит в `t`:

`c:=pos('ом','компьютер');` (`c=2`).

В Паскале определены также четыре стандартные процедуры для обработки строковых величин:

1. Процедура удаления `delete(s,i,k)`. Из строки `s` удаляется `k` символов, начиная с `i`-того символа.

`s:='таракан'; delete(s,5,2);` (в результате `S='таран'`).

2. Процедура вставки - `insert(s,t,i)`. Строка `s` вставляется в строку `t`, начиная с позиции `i`:

`t:='таран'; insert ('ка',t,5);` (`t='таракан'`).

3. Процедура преобразования числа в строку символов - `str(k,s)`. Строка `s` получается «навешиванием» апострофов на число `k`:

```
str(564,s); (s='564').
```

4. Процедура преобразования строки из цифр в число - val(s,k,i). Число i=0, если в строке s нет символов, отличных от цифр, в противном случае i=позиции первого символа, отличного от цифры:

```
val('780',k,i); (k=780; i=0).
```

Рассмотрим несколько программ, в которых используются строковые величины.

1. Составить программу, определяющую количество гласных в русском тексте, содержащем не более 100 символов.

Здесь удобно определить констант-строку, состоящую из всех 18 строчных и заглавных русских букв, и в цикле проверить, будет ли очередной символ заданного текста элементом констант-строки.

Программа 11

```
program vowel;  
const c:string[18]='аеиоуыэюяАЕИОУЫЭЮЯ' ;  
var a :string[100]; k,n:integer;  
begin  
    writeln('введите текст'); readln(a);n:=0;  
    for k:=1 to length(a) do  
        if pos(a[k],c)>0 then n:=n+1;  
        writeln('кол. гласных=',n) end.
```

2. Заменить в арифметическом выражении функцию sqr на exp. Замена выражения sqr на exp достигается последовательным применением процедур delete и insert:

Программа 12

```
program stroka;  
    var a,b:string[40]; k:integer;  
begin  
    writeln('введите строку <= 40 символов');  
    readln(a);b:=a;  
    repeat k:=pos('sqr',b);  
        if k>0 then  
            begin  
                delete(b,k,3);insert('exp',b,k) ;  
            end  
    until k=0;  
    writein('старая строка=',a); writein('новая строка"',b);  
end.
```

3. Ввести и упорядочить по алфавиту 10 латинских слов. В программе определим массив из 10 элементов-строк и упорядочим его элементы методом пузырька.

Программа 13

```
program order;  
const s=10;  
type word=string(20) ;  
var i, j, k : 1..s;  
    b:word; p:boolean; list :array[1..s] of word;  
begin  
    clrscr;writeln('введите список слов');  
    for i:=1 to s do readln(list[i]);  
    repeat p:=true;  
    for i:=s downto 2 do  
        if list[i]<list[i-1] then
```



```

begin
  b:=list[i];list[i]:=list[i-1]; list[i-1]:=b;p:=false end
until p=true;
writein('упорядоченный список слов:');
  for i:=1 to s do writeln(list[i])
end.

```

Множество в Паскале имеет такой же смысл, как и в алгебре - это неупорядоченная совокупность отличных друг от друга однотипных элементов. Число элементов множества не должно превышать 255. В качестве типа элементов может быть любой скалярный тип, кроме типа integer и его интервалов, содержащих числа > 255 . Тип элементов множества называется базовым. При описании множественного типа используются служебные слова set и of. Задание конкретного множества определяется правилом (конструктором) - списком элементов или интервалов, заключенным в квадратные скобки. Пустое множество обозначается двумя символами [].

Множественный тип можно определить в разделе описания типов по схеме:

```
type <имя> = set of <тип элементов>
```

Например:

```
type t=set of byte;
var a:t;
```

Можно совместить описание множественного типа и соответствующих переменных:

```
var code: set of 0..7;
digits: set of '0'..'9';
```

Можно описать переменную множественного типа и задать ее первоначальное значение в разделе описания констант, как константмножество. Тип множества можно описать ранее, например,

```
type up=set of 'A'..'Z';
low=set of 'a'..'z";
const upcase : up=['A'.. 'Z'];
vocals :low=['a', 'e', 'i', 'o', "u", 'y'];
delimiterset of char=[ '..' /',' : '..' ? ' '];
```

Для данных множественного типа определены операции объединения, пересечения и дополнения множеств, обозначаемые в Паскале соответственно знаками +, * и -, а также отношения равенства множеств ($A=B$), неравенства ($A \neq B$), включения ($A \subseteq B, A \supseteq B$).

Логическая операция принадлежности $x \in A$ принимает значение true, если элемент x принадлежит множеству A и false в противном случае. Так как к элементам множества прямого доступа нет, то операция in часто используется для этой цели.

Заметим, что операции отношения на множествах выполняются быстрее, чем соответствующие операции на числах, поэтому их выгодно применять в программах.

Пример: составить программу, анализирующую латинский текст и печатающую в алфавитном порядке все найденные в нем буквы, а затем все найденные.

Пусть alfa - множество всех букв латинского алфавита. Будем вводить заданный текст с клавиатуры символ за символом, одновременно формируя множество E -множество латинских букв текста. В конце текста введем символ *. Затем с помощью операции in будем проверять, какие буквы алфавита имеются во множестве E. Множество N - найденных букв в тексте - определяется оператором: $N := alfa - E$.

```

program search;
const alfa:set of char=['a' . . 'z'];
var c:char;E,N:set of char;
begin
    cirscr; E:=[]; writeln('введите текст, конец ввода -*'); read(c);
while c<> '*' do
begin
    if c in alfa then E:=E+[c]; read(c)
    end;
writeln;
if E=alfa then writeln('найжены все латинские буквы')
else begin
    N:=alfa-E;
writeln('найжены:');
    for c:='a' to 'z' do if c in E then write(c);
writeln; writeln('не найжены:');
    for c:='a' to 'z' do if c in N then write(c);
    writeln
    end
end.

```

Переменные множественного типа удобно применять в задачах, где порядок данных не имеет значения, например при моделировании случайных событий.

Пример: составить программу «спортлото 5 из 36», которая позволяет человеку ввести с клавиатуры пять натуральных чисел из интервала 1..36, затем генерирует случайным образом пять различных чисел из того же интервала и объявляет величину выигрыша по правилу: если угаданы человеком 0, 1 или 2 числа, объявляется проигрыш; если угаданы 3 числа, объявляется выигрыш 3 рубля; если угаданы 4 числа, объявляется выигрыш 100 рублей; если угаданы 5 чисел, объявляется выигрыш 1000 рублей.

В программе используются обозначения: mn — множество натуральных чисел из интервала 1..36, a - множество чисел, задуманных человеком, x - множество чисел, генерируемых компьютером, $z=a*x$ - пересечение множеств a и x; i, k, s - переменные, значения которых принадлежат интервалу 1..36. Случайное число из этого интервала генерируется оператором: $s:=\text{random}(35)+1$. Программа сначала выводит на экран сообщение о выигрышных номерах, затем определяет величину выигрыша.

Программа 15

```

program lottery;
type mn = set of 1 . . 36;
var x,a, z: mn; i, k, s: 0 . . 36;
begin
    writeln; a:=[];
    for i:=1 to 5 do
        begin write('введите ',i,' -тое число '); readln(k);a:=a+[k]
        end;

    randomize;
    k:=0; x:=[];
while k<5 do
begin
    s:=random (35) +1;
    if not(s in x) then begin k:=k+1; x:=x+[s]
    end
end; writeln;
writeln('выигрыш выпал на следующие номера : ');
for i:=1 to 36 do if (i in x) then write(i,' ');
writeln; z:=a*x; k:=0;
for i:=1 to 36 do if (i in z) then begin
writeln('угадано: ',i); k:=k+1

```

```

        end;
    case k of
        0, 1, 2 : writeln('вы проиграли ');
        3 : writeln('получите 3 руб') ;
        4 : writeln('получите 100 руб');
        5 : writeln('получите 1000 руб')
    end
end.

```

Записи (комбинированный тип) - одна из наиболее гибких и удобных структур данных, применяющихся при описании сложных объектов, которые характеризуются различными свойствами, а также при создании различных информационных систем. Запись - это последовательность, состоящая из фиксированного числа величин разных типов, называемых полями или компонентами записи. Так же, как и массив, запись содержит ряд отдельных компонент, но компонентами записи могут быть данные различных типов. Например, адресные данные (индекс, город, улица, номер дома, квартиры) можно представить как запись (record):

```

type address = record
    index : string[6];
    city: string[20];
    street : string[20];
    haus.ilat: integer
end;

```

Из примера видно, что тип «запись» описывается по схеме

```

type имя типа записи = record
    имя поля 1 : тип;
    имя поля 2 : тип;
    имя поля N: тип
end;

```

Как и при описании массивов, можно совместить описание типа записи и соответствующих переменных. Например, данные о двух студентах можно описать так:

```

fio      :string[20];
fas      :string[10]
grup     :string[8]
end

```

Переменную типа «запись» и ее первоначальное значение можно определить как констант-запись в разделе констант по схеме

```

const <имя>: <имя типа> = <константное значение>

```

Константное значение - это список имен полей и соответствующих значений, заключенный в круглые скобки. Элементы списка разделяются знаком «точка с запятой». Например, запись о начале координат можно определить так:

```

type point = record
    x, y, z: integer
end;
const o: point = (x:0; y:0; z:0);

```

С компонентами записи можно обращаться как с переменными соответствующего типа. Обращение к компонентам записи осуществляется с помощью указания имени поля через точку.

Пусть, например, переменная *x* имеет тип *address*, т.е. в программе имеется описание *var x: address*. Тогда допустимы следующие присваивания:

```
x.haus := 52; x.street:='пр.Мира'; x.city:= 'Красноярск';  
x.flat:= 135; x.index :='660049'
```

Проиллюстрируем работу с записями на задаче, в которой требуется найти сумму и произведение двух комплексных чисел:

```
z1=a1+i*b1; z2=a2+i*b2.
```

Программа 16

```
program cornpl;  
type compi = record  
    re : real;  
    im : real  
end;  
var z1,z2,s,p : compi;  
begin  
    writeln('компл.число a+i*b вводите двумя числами a и b: ');  
    write('введи 1 число: '); readln(z1.re,z1.im);  
    write('введи 2 число: '); readln(z2.re,z2.im);  
    s.re := z1.re + z2.re;  
    s.im := z1.im + z2.im;  
    p.re := z1.re * z2.re - z1.im * z2.im;  
    p.im := z1.re * z2.im + z2.re * z1.im;  
    writeln('s=',s.re:4:2,' + i * ',s.im:4:2);  
    write('p=',p.re:4:2,' + i * ',p.im:4:2);  
end.
```

Громоздкость обозначений в программе компенсируется большей наглядностью алгоритма за счет структуризации данных. Во многих случаях, если требуется производить операции с полями фиксированной записи, можно для сокращения обозначений использовать оператор присоединения *with*. Его структура такова:

```
with <имя записи> do <оператор>;
```

В этом случае в операторе, написанном после служебного слова *do*, имена полей указанной записи описываются без имени записи и точки. Например, печать суммы *s* в предыдущем примере можно организовать с использованием оператора *with* так:

```
with s do writeln('s=',re:4:2,'+i*',im:4:2);
```

В операторах присваивания разрешается использовать не только имена полей, но и имена записей.

Тип поля может быть записью.

Например:

```
man = record  
    fio:record  
        fam, im, otch : string[10];  
    end;  
    data : record  
        day: 1..31;  
        mes:1..12;  
        god:integer
```

```

end;
pol: char;
telef: record
dom,rab : string[10];
end;
end;

```

В Паскале разрешается использовать тип «запись» при описании других составных типов данных, например, можно построить массив записей.

Рассмотрим пример эффективного использования записей в программе начисления стипендии студентам по шаблону:

N	ФИО	Эк1	Эк2	Эк3	Балл	Сумма	Проф	Итого
1	Васнецов Н.В.	4	4	3	11	50.00	0.25	49.75

Предположим, что вводится список группы с соответствующими оценками за экзамены. Графа «Балл» вычисляет суммарную оценку за семестр. Графа «Сумма» определяет размер стипендии по упрощенному правилу: если нет двоек и балл равен 15, то стипендия 75 руб.; при условии, что $12 < \text{«Балл»} < 15$ стипендия 62 руб 50 коп., а если $9 < \text{«Балл»} < 12$, то - 50 руб. (в других случаях сумма равна нулю). В графе «Проф» указывается профсоюзный взнос в размере 0,5% от стипендии, а графа «Итого» определяет сумму денег к выдаче.

В программе перед распечаткой итоговой ведомости можно предусмотреть упорядочение записей по убыванию в графе «Балл».

Программа 17

```

program spisok;
type stud = record
    fio           :string[20];
    ex1, ex2, ex3 : 2 . . 5;
    bal           :6 . . 15;
    sum           :real;
    nalog         :real;
    itog          :real;
end;
var x : array[1..30] of stud;
    i,k,m,n      :integer;
    y            : 6..15;
    z            : stud;
begin write('введи число студентов: '); readln(n);
    for i:= 1 to n do with x[i] do
        begin write('введи ФИО ',i,'-го студента: ');
            readln(fio); write('Введи его три оценки: ');
            readln(ex1,ex2,ex3);
        end;
    for i:= 1 to n do with x[i] do
        begin bal:=ex1+ex2+ex3;
            if (ex1=2) or (ex2=2) or (ex3=2)
                then sum:=0
            else if bal=15 then sum:=75
            else if bal>12 then sum:=62.5
            else if bal>9 then sum:=50
            else sum:=0;
            nalog:=sum*0.005; itog:=sum-nalog;
        end;
    for k:= 1 to n-1 do
        begin y:=x[k].bal; m:=k;
            for i:=k+1 to n do if y<x[i].bal then
                begin y:=x[i].bal; m:=i

```

```

        end; z:=x[k]; x[k]:=x[m]; x[m]:=z;
    end;
    writeln; writeln ('СТИПЕНДИАЛЬНАЯ ВЕДОМОСТЬ ');
    for i:=1 to 64 do write('-'); writeln;
    write ('N | ФИО | эк1 | эк2 | эк3 | балл | сумма | проф | итого |') ;
for i:=1 to 64 do write('-'); writeln;
for i:=1 to n do with x[i] do
    begin write(i:3,fio:20,exi:4, ex2:4,ex3:4);
    writeln(bal:5,sum:9:2,nalog:8:2,- itog:7:2);
    end
end.

```

Контрольные вопросы и задания

1. Как определяется перечислимый тип данных?
2. Для чего может понадобиться интервальный тип данных?
3. Как вводятся и используются в программах массивы?
4. Какие действия возможны над величинами строкового типа?
5. Какие операции допустимы над множествами?
6. В чем принципиальные различия между одномерными массивами и записями?
7. В усеченном конусе длина диагонали осевого сечения равна d , образующая составляет с плоскостью основания угол α и равна a . Вычислите площадь боковой поверхности конуса.
8. Вычислите объем призмы, боковые грани которой квадраты, а основанием служит равно-сторонний треугольник, вписанный в круг радиуса r .
9. Числа a и b выражают длины катетов одного прямоугольного треугольника, числа c и d - другого. Определите, являются ли треугольники подобными.
10. Напечатайте числа a, b, c в порядке возрастания.
11. Определите все пары двузначных чисел, обладающих свойством: $(20+25^2 = 2025)$.
12. Вычислите в числовом массиве $a_1, a_2, \dots, a_n$ суммы положительных и отрицательных элементов.
13. Вычислите скалярное произведение двух десятимерных векторов X и Y .
14. Упорядочите массив $x_1, x_2, \dots, x_n$ по неубыванию, используя метод сортировки вставками: пусть первые k элементов уже упорядочены по неубыванию; берется $(k+1)$ -й элемент и размещается среди первых k элементов так, чтобы упорядоченными оказались уже $(k+1)$ первых элементов.
15. Составьте программу решения треугольной системы уравнений порядка n .
16. Замените в заданном арифметическом выражении все вхождения $\sin$ на $\cos$ и $\sqrt{x}$ на abs .
17. Для заданного текста определите длину содержащейся в нем максимальной последовательности цифр $0, 1, 2, \dots, 9$.
18. Дан текст из латинских букв и знаков препинания. Составьте программу частотного анализа букв этого текста, т.е. напечатайте каждую букву с указанием количества ее вхождений и процента вхождений.
19. Найдите и напечатайте в порядке убывания все простые числа из диапазона $[2..201]$.
20. Опишите тип «запись» для следующих данных:
 а) адрес(город, улица, дом, квартира);
 б) дата(число, месяц, год);
 в) студент(фio, факультет, курс, группа).
21. Заданы N точек на плоскости. Найдите точку, ближайшую к началу координат. Используйте тип «запись».

3.4. ПРОЦЕДУРЫ И ФУНКЦИИ

Описание и вызов. В Паскале подпрограммы называются процедурами и функциями и описываются в разделе с тем же названием.

Процедура имеет такую же структуру, как и программа, но с двумя отличиями:

- заголовок процедуры имеет другой синтаксис и включает служебное слово `procedure`;

- описание процедуры заканчивается точкой с запятой (а не точкой). Все имена, описанные в программе до процедуры, действуют во всей программе и в любой ее подпрограмме (если они там не описаны заново). Они называются глобальными, в отличие от локальных имен, описанных в процедуре и действующих лишь в ней.

Данные для обработки могут передаваться процедуре через глобальные имена или через аргументы процедуры. В процедуре каждый аргумент имеет свое имя -формальный параметр, описываемый в заголовке процедуры по схеме

procedure <имя> (<список описаний формальных параметров>) Описание формальных параметров может иметь вид
<список имен>: <тип> или var <список имен>: <тип>

В первом случае говорят о параметрах-значениях, во втором - о параметрах-переменных. В простейшем случае заголовок процедуры может содержать только имя процедуры.

Оператор вызова процедуры имеет вид

<имя процедуры> (<список выражений>);

Указанные выражения называют фактическими параметрами. Их список должен точно соответствовать списку описаний формальных параметров процедуры. Во время вызова процедуры каждому параметру-значению присваивается значение соответствующего фактического параметра и поэтому их обычно используют для передачи входных данных.

Параметры-переменные следует использовать для представления результатов процедуры.

Пример: составим программу, которая с помощью строки символов разделит экран на части, где напечатает таблицу квадратных корней для чисел 1, 2,..., 10 и таблицу натуральных логарифмов для чисел 1, 2,..., 5.

Печать строки символов оформим как процедуру. Так как никакую информацию передавать из процедуры в программу не надо, то аргументы процедуры (вид и количество символов) будут описаны как параметры-значения.

Заметим, что процедура в программе выполняется пять раз.

Программа 18

```
program section;
var x:integer;
    procedure line(a:integer;c:char) ;
    var j:integer;
    begin
        for j:=1 to a do write (c);
        writeln
    end;
begin
    line(35,'-'); writeln('таблица квадратных корней');
    line(35,'-');
    for x:=1 to 10 do writeln(x:8,sqrt(x):8,4);
    line (35,'-'); writein('таблица натуральных логарифмов');
    line(35,'-');
    for x:=1 to 5 do writein(x:8,ln(x):8:4);
    line(35,'*')
end.
```

Функция - это подпрограмма, определяющая единственное скалярное, вещественное или строковое значение. Отличия подпрограммы-функции от процедуры:

- заголовок функции начинается со служебного слова function и заканчивается указанием типа значения функции:

function <имя> (список описаний формальных параметров): <тип>;

•раздел операторов функции должен содержать хотя бы один оператор присваивания имени функции;

• обращение к функции - не оператор, а выражение вида

<имя функции> (<список фактических параметров>).

Функции (и процедуры) могут использовать свое имя в собственном описании, т.е. могут быть рекурсивными.

Пример: составим программу, которая для заданных четырех натуральных чисел a, b, c, d напечатает наибольшие общие делители первой и второй пар чисел и сравнит их по величине.

В программе определим **рекурсивную функцию** nod(x,y) по формулам

$$\begin{aligned} &| x, \text{ если } y = 0 \\ \text{nod}(x,y) &= | \text{nod}(y,x), \text{ если } x < y \\ &| \text{nod}(x \bmod y, y), \text{ если } x > y \end{aligned}$$

Применяя эти формулы к числам 21 и 15, последовательно находим $\text{nod}(21,15) = \text{nod}(6,15) = \text{nod}(15,6) = \text{nod}(3,6) = \text{nod}(6,3) = \text{nod}(0,3) = \text{nod}(3,0) = 3$.

Программа 19

```
program four;
  var a,b,c,d,m,n:integer;
  function nod(x,y:integer):integer;
  var h:integer;
begin
  if y=0 then h:=x
    else if x<y then h:=nod(y,x)
    else h:=nod(x mod y, y);
nod:=h end;
begin
  writeln('введите 4 натуральных числа');
  read(a,b,c,d); writeln;
  m:=nod(a,b); n:=nod(c,d);
  writeln('нод(' , a , ', ' , b , ')=' , m);
  writeln('нод(' , c , ', ' , d , ')=' , n);
  if m>n then writeln('первый > второго')
    else if m<n then writeln('первый < второго')
    else writeln('нод пар равны') end.
```

Внешние библиотеки. Как известно, подпрограммы (процедуры и функции) используются в программах с целью их структурирования, а также при многократных повторениях некоторых частей программы. Процедуры и функции описываются в программных единицах в разделе описания подпрограмм. Они являются внутренними для этих программных единиц.

Бывают случаи, когда одни и те же подпрограммы могут использоваться в различных программах одного и даже нескольких пользователей. В подобных ситуациях целесообразно создавать внешние подпрограммы, которые можно в необходимый момент подключать в любые программы. Как правило, внешние подпрограммы объединяют в отдельные пакеты, так называемые, библиотеки внешних подпрограмм. Могут создаваться личные библиотеки, специализированные библиотеки коллективного пользования и др. С одной из таких библиотек - встроенной библиотекой стандартных подпрограмм - пользователи имеют дело практически всегда. В состав этой библиотеки входят процедуры и функции вычисления значений ряда элементарных функций: синуса, косинуса, экспоненты и т.д., процедуры и функции обработки символьных величин, процедуры ввода-вывода и др. (список их приведен в конце § 3). Встроенная библиотека подключается к любой программе автоматически при компиляции. Поэтому откомпилированный файл с расширением .com (иногда называемый «комовским»), как правило, занимает в 8 -10 раз больше места в памяти, чем исходный текст.

Внешние подпрограммы создаются обычным образом в виде отдельного файла или файлов.

Для подключения внешних подпрограмм в программе пользователя в разделе описания ставится директива \$I имя файла. С этого момента все процедуры и функции внешнего файла становятся внутренними для программы, и на все входящие в него процедуры и функции распространяется правило локальных и глобальных переменных. В этой связи, директива подключения внешнего файла должна размещаться после описания всех ею используемых глобальных параметров, процедур и функций.

Пример. Создадим внешнюю библиотеку из двух процедур и одной функции. Первая процедура программы 20 очищает экран, выдает приветствие, затем после нажатия клавиши <Пробел> снова очищает экран. Вторая процедура возводит число a в степень b. Третья подпрограмма-функция вычисляет значение экспоненты с некоторым грубым приближением на основе ряда Тейлора.

Программа 20

```
procedure PRIVET;
var a: char;
begin
    cirscr; gotoxy(20,10) ;
    write('здравствуйте , желаю успехов !') ;
    repeat (цикл позволяет)
        gotoxy(35,50);write('пробел'); (сменить экран)
        read(kbd,a); (по нажатию клавиши)
        until a=' '; (* 'пробел' )
        cirscr;
    end;
    procedure STEPEN(a,b:real;var y:real);
    begin
        y:=exp(b*ln(a)) ;
    end;
    function MEXP(x:real):real;
    begin
        mexp:=1+x+x*x/2+x*x*x/6+x*x*x*x/24;
    end;
```

Пусть представленные три подпрограммы записаны в файл с именем lab.pas. А теперь составим программу, использующую созданную внешнюю библиотеку.

Программа 21

```
program primeri;
($i lab) (директива подключения библиотеки) var a,b : real;
begin
    PRIVET;
    STEPEN(2,4,a); writeln('2 в степени 4 =',a); b:=MEXP(1);
    write('машинная exp(1)=' ,EXP(1):6:4, ' моя exp(1)=' ,b:6:4);
end.
```

В программе используется стандартная функция - экспонента EXP(1) и наша подпрограмма MEXP(1).

Модули используют в более поздних версиях Паскаля для создания библиотек и разделения больших программ на логически связанные независимые друг от друга составные части. В состав модуля входят следующие разделы: заголовок, интерфейс, реализация, инициализация. Заголовок необходим для ссылок на модуль. Интерфейс содержит объявления, включая процедуры и функции, представленные списком заголовков и доступные пользователям в теле основной программы. Раздел «реализация» содержит тела процедур и функций, перечисленных в интерфейсной части модуля. Раздел «инициализация» содержит операторы, необходимые для инициализации модуля. Таким образом модуль - это набор констант, типов данных, переменных, процедур и функций. Каждый модуль компилируется отдельно; результат компиляции - файл с расширением .tpu (Turbo Pascal Unit). Каждый элемент модуля можно использовать в программе пользователя без дополнительного объявления, для чего достаточно записать имя используемого модуля в ди-

рективе Uses в начале программы после его заголовка.

В Турбо-Паскале версии 5.0 и выше применяют стандартные модули CRT, GRAPH и др. В этих модулях содержатся сервисные процедуры и функции по работе с экраном дисплея, с клавиатурой, графическими примитивами и т.п. Модули подключаются к программе путем специальной команды, размещаемой сразу после заголовка:

```
uses <имя модуля>
```

Программист может сам создать модуль. Ниже приведен пример с соответствующими комментариями.

Пример. Создать модуль, дополняющий математические возможности Паскаля арифметическими действиями над комплексными числами.

Будем представлять комплексные числа парами действительных: (a, b). Как известно, действия над ними выполняются по правилам

$$\begin{aligned}(a,b) + (c,d) &= (a+c,b+d), \\ (a,b)-(c,d) &= (a-c,b-d), \\ (a,b) * (c,d) &= (a*c-b*d, a*d+b*c), \\ (a,b) / (c,d) &= ((a*c+b*d)/(c*c+d*d), (b*c-a*d)/(c*c+d*d)).\end{aligned}$$

Создаваемый модуль будет включать четыре процедуры: Sum - сумма, Raz -разность, Proiz - произведение, Chstn - частное.

Этот модуль может быть отдельно откомпилирован. После этого любая программа, написанная на Паскале, может получить доступ к интерфейсным объектам (в данном случае - процедурам) этого модуля с помощью директивы Uses CompChisla.

Обратим внимание, что в интерфейсной части модуля от процедур присутствуют лишь заголовки, а в части «реализация» от заголовков процедур остаются лишь их имена.

Программа 22

```
unit CompChisla;
interface
    procedure Sum(a,b,c,d: real; var x,y: real);
    procedure Raz(a,b,c,d: real; var x,y: real);
    procedure Proiz(a,b,c,d: real; var x,y: real);
    procedure Chstn(a,b,c,d: real; var x,y: real);
implementation
    procedure Sum;
    begin x:=a+c; y:=b+d end;
procedure Raz;
begin x:=a-c; y:=b-d end;
procedure Proiz;
begin x:=a*c-b*d; y:=a*d+b*c
end;
procedure Razn;
var z:real;
begin z:= c*c+d*d; x:=(a*c+b*d)/z; y:=(b*c-a*d)/z end;
end.
```

Контрольные вопросы

1. Какова структура процедуры? функции?
2. Какие параметры называют формальными и какие - фактическими?
3. В чем различие между локальными и глобальными переменными?
4. В чем сходство и различие между процедурой и модулем?

3.5. РАБОТА С ФАЙЛАМИ

Файл (последовательность) - это одна из наиболее фундаментальных структур данных. Программная организация компьютеров, их связь с внешними устройствами основаны на файловой структуре.

Файлы позволяют решить две проблемы:

1) возможность формирования и сохранения значений для последующего использования другими программами (например, в программах многократной обработки информационных систем, таких как платежные ведомости, различные АСУ, базы данных, необходимость длительного хранения информации очевидна);

2) взаимодействие программ с внешними устройствами ввода-вывода: дисплеем, принтером, АСП и т.п.

В Паскале эти проблемы снимаются с помощью структурированных данных файлового типа.

Файловый тип данных в программе задается следующим образом:

```
type <имя файлового типа> = file of <тип компонентов>
```

В качестве типа компонентов файла разрешается использовать любой тип данных, кроме файлового. Например:

```
type
intfile = file of integer;
refile = file of real;
chfile = file of char;
ran = 1 .. 10;
st = set of ran;
vector = array[ran] of real;
comp = record
re,im : integer;
end;
setfile=file of st;
vecfile = file of vector;
compfile= file ofcomp;
```

Описание файловой переменной задается обычным способом в разделе описаний. Например:

```
var f: intfile; или var f: file of integer;
```

Файловая переменная является буфером между Паскаль-программой и внешним устройством и должна быть логически с ним связана. Связь осуществляется оператором языка Паскаль:

```
assign (<имя файловой переменной>,<имя устройства> ')
```

Как правило, файлы для хранения данных связаны с устройством внешней памяти на магнитных носителях (дисковод) и носят название **внешние файлы**. Если, например, файл с именем primer.dat логически связан с дисководом А:, то все данные, помещаемые в файл, будут храниться на этом дисковом накопителе, а установка «окна» между программой и файлом будет определяться через файловую переменную f оператором

```
assign (f, 'primer.dat')
```

Если внешним устройством является принтер, то связь осуществляется оператором assign(f, '1st:'). Здесь 1st - логическое имя печатающего устройства. Ниже приведены логические имена внешних устройств ввода-вывода:

con - консоль; trm - терминал; kbd - клавиатура; lst - принтер; aux - буфер сети;
usr-драйвер пользователя.

После осуществления связи файловая переменная *f* отождествляется с соответствующим файлом.

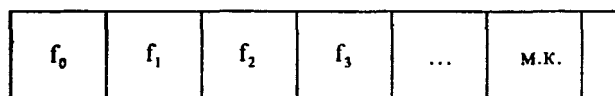
Для работы с файлом его необходимо открыть, а по окончании работы — закрыть. Файл открывается для чтения оператором `reset(f)`, для записи - оператором `rewrite(f)`.

Чтение и запись данных осуществляется известными командами `read/write`, только в начале списка помещается имя файловой переменной:

```
read (f, <список ввода>); readln (f, <список ввода>);  
write(f, <список вывода>); writeln(f, <список вывода>).
```

Заккрытие файла осуществляется командой `close(f)`.

УСЛОВНО файл можно представить в виде ленты, у которой есть начало, а конец не фиксируется. Компоненты файла записываются на эту ленту последовательно, друг за другом:



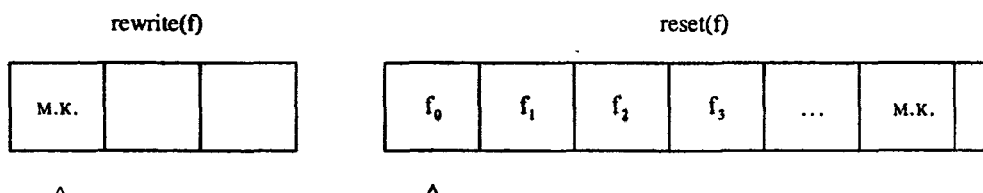
^ т.м.

Здесь т.м. - текущий маркер, указывающий на рабочую позицию (окно) файла; м.к. (маркер конца файла) - специальный код, автоматически формируемый вслед за последним элементом файла.

Такого рода файлы называются **файлами последовательного доступа**. В исходной версии Паскаля **файлов прямого доступа**, для которых можно непосредственно «достать» любую компоненту, не предусмотрено; однако, в Турбо-Паскале элементы прямого доступа есть (например, через функцию `seek`, см. ниже).

Команда `rewrite(f)` - открыть файл для записи - устанавливает файл в начальное состояние режима записи; текущий маркер устанавливается на маркер конца файла. Если в файле *f* до этого была информация, то она уничтожается.

В открытом для чтения командой `reset(f)` файле текущий маркер устанавливается на нулевое состояние, однако содержимое файла не утрачивается.



Команда закрытия файла `close` (Q обязательна, поскольку эта команда формирует маркер конца файла, что в большинстве случаев является необходимым например, для работы с функцией `eof(f)`, см ниже).

В системе Турбо-Паскаль предусмотрены встроенные функции по работе с файлами:

`filesize(f)` - текущее количество компонент открытого файла;

`filepos(f)` -номер текущей позиции маркера;

`rename (f,имя)` - переименование файла, связанного с *f*;

`erase(f)` -уничтожение файла;

`execute(f)` - выполнение COM-файла;

`chain(f)` -выполнение CHN-файла;

`seek(f,N)` - устанавливает маркер на позицию *N*;

`eof(f)` - возвращает TRUE, если найден конец файла;

`ealn(f)` - возвращает TRUE, если найден конец строки.

На практике широко используются **текстовые файлы**, которые состоят из литерных (логи-

ческих) строк. Поэтому в языке Паскаль предусмотрен стандартный файловый тип TEXT (он не является file of char, скорее всего, это - file of string[n]). Логические строки бывают разной длины, в том числе и нулевой. В конец каждой строки помещается специальный символ «конец строки» (eoln - «end of line»). В качестве печатного символа конца строки используют литеру #. Текстовый файл (text) является строго последовательным, к нему не применимы некоторые встроенные функции, в частности, seek. В отличие от типизированных файлов, с текстовым файлом нельзя одновременно проводить операции чтения (read) и записи (write). Однако, допустимы операторы writein и readln. Числовые данные, целые и вещественные, в текстовом файле должны записываться через пробел.

Ниже приведены несколько примеров, иллюстрирующих работу с файлами. *Пример 1.* Вывод данных на печатающее устройство - принтер (lst:).

Программа 23

```
program print;
var
  fal :text; x :real; name :string[25];
begin
  assign (fal, ' lst:'); rewrite (fal); x:2.5; name: "'Слава';
  writeln(fal,x:8:2) ;
  writeln(fal,'Привет, '.name); close(fal) :
end. (Здесь файловая переменная fal связывается с принтером lst:, и запись
      в файл fal практически означает вывод на печать)
```

Пример 2. Создание и сохранение в файле «xxx.dat» последовательности целых чисел от 10 до 20.

Программа 24

```
program zapis;
var
  f: file of integer; i: integer;
begin
  assign(f, 'xxx.dat'); rewrite(f);
  for i:=10 to 20 do write(f,i); close (f);
end. (После выполнения программы формируется внешний файл xxx.dat)
```

Пример 3. Считывание первых пяти компонент из существующего файла «xxx.dat» и вывод на дисплей квадратов этих значений.

Программа 25

```
program read;
var ff: file of integer; j,i : integer;
begin
  assign(ff, 'xxx.dat'); reset(ff) ;
  for j:=1 to 5 do begin read(ff,i); writeln(i*i);
end;
close(ff);
end.
```

Пример 4. В текстовом файле (text) «slov.txt» содержится русский текст. Определить сколько гласных букв в тексте.

Программа 26

```
program text;
var
  ft : text; n : integer; ch : char; st : set of char begin
```

```

    assign (ft,'slov.txt'); reset(ft);
st := ['А','Е','И','О','У','Ы','Э','Ю','Я'];
st := st+['а','е','и','о','у','ы','э','ю','я'];
n:=0;
while not eof(ft) do begin
    read(ft, ch); if ch in st then n:=n+1;
end;
close(ft);
    writeln; write('кол-во гласных букв =',n);
end.

```

Поскольку длина текста (файла) неизвестна, то в цикле «пока» используется логическая функция eof(f), которая возвращает значение TRUE, если найден конец файла.

Пример 5. Шифрование и дешифрование текста.

Программа шифрования заданного текста (sekret) использует следующее правило шифровки: каждая буква в тексте заменяется на букву, расположенную на n позиций вправо от искомой в русском алфавите. Причем последним символом алфавита является пробел " ", а далее алфавит продолжается циклически.

Значение смещения n находится во внешнем файле 'n.key', который формируется программой key. Зашифрованный текст выводится во внешний файл с именем «шифр.ехе» , а также на дисплей.

Программа дешифровки (getsek) считывает зашифрованный текст из файла «шифр.txt» и выводит на экран дисплея искомый текст.

Программа 27 (а)

```

program key;
var
    n : integer; f : file of integer;
begin , .
assign (f,'n.key'); rewrite(f); write\('Введи ключ(смещение): ');
readln(n);
    write(f,n); close(f);
end.

```

Программа 27 (б)

```

program sekret;
var
| slovo,anslovo: string[100];
alfavit : string[33];
n, i, k, p : integer;
fkl : file of integer;
fs : text;
begin
    alfavit:='абвгдежзийклмнопрстуфхцщъыьэюя ';
    assign(fkl,'n.key'); reset(fkl); read(fkl,n); close(fkl);
    writeln; write('введи текст: ');
    readln(slovo); anslovo:='';
    for k:=1 to length(slovo) do begin for i:=1
to 33 do
        if slovo[k]=alfavit[i] then begin p:=i+n;
            if p >33 then p:=p mod 33;
            anslovo:=anslovo+alfavit[p];
        end;
    end;
    assign(fs,'шифр.txt'); rewrite(fs); write(fs,anslovo);close(fs);
    writeln; write(anslovo) ;
end.

```

Программа 27 (в)

```
program retsek;
var slovo, anslovo : string[100];
    alfavit : string[33];
n, i, k, p : integer;
fi : file of integer;
f : text;
begin alfavit:='абвгдежзийклмнопрстуфхцчшщъыьэюя ';
    assign(fi,'n.key'); reset(fi); read(fi.n); close(fi);
        assign(f,'шифр.txt'); reset(f); read(f,anslovo); close(f)
    slovo:=' ' ;
    for k:=1 to length(anslovo) do
    begin for i:=1 to 33 do
        if anslovo[k]=alfavit[i] then
            begin p:=i-n; if p < 1 then p:=33-p mod 33;
            slovo:=slovo+alfavit[p];
            end;
        end;
    end;
    writeln; write('Текст шифровки: ', slovo);
end.
```

Контрольные вопросы и задания

1. Какие проблемы решаются при использовании файлов?
2. Какие операции и функции используются при работе с файлами?
3. Найдите площадь выпуклого четырехугольника со сторонами x , y , z , t и одной из диагоналей d .
4. Заданы три комплексных числа. Найдите их сумму,
5. Имеется внешний файл записей «lab.zap», содержащий сведения об учениках школы.

Файл формируется приведенной ниже программой.

Составьте программу, в которой необходимо сделать следующее:

- а) упорядочить файл по признаку «class» в порядке возрастания;
- б) упорядочить файл по признаку «god» в порядке убывания;
- в) упорядочить файл в алфавитном порядке.

```
program lab;
type shcoo 1= record
fio : string[20], class : 1 .. 11; god : integer; pol: char;
end;
var x:array[1..100] of shcoo; n,i:integer; f:file of school;
begin
write('введите кол-во учеников:'); readln(n); assign(f,'lab.zap'); rewnt(f);
for i:=1 to n do with x[i] do
begin \write('введи Ф.И.О.',1,'-ого ученика:'); readln(fio);
write('класс:'); readln(class);
write('год рождения:'); readln(god); write('пол(М/Ж):-');
readln(pol); write(f,x[i]);...;
end; close(f);
end.
```

6. Задание рассчитано на двух студентов, использующих «электронную почту» на одном компьютере. Первый студент составляет программу, в которой формирует внешний файл «письмо». Второй студент должен «прочитать» файл и сформировать «ответ».

ПИСЬМО

- а) последовательность целых чисел до 100
- б) простые числа в интервале от 1 до 200

ОТВЕТ

- | |
|----------------------|
| квадраты этих чисел; |
| сумма этих чисел |

3.6. ДИНАМИЧЕСКИЕ ИНФОРМАЦИОННЫЕ СТРУКТУРЫ

Динамические переменные и указатели. До сих мы рассматривали **статические** переменные. Такие переменные автоматически порождаются при входе в тот блок, в котором они описываются, существуют на протяжении работы всего блока и уничтожаются при выходе из этого блока.

Обращение к статическим переменным производится по их именам, а тип определяется их описанием. Вся работа по размещению статических объектов в памяти машины выполняется на этапе трансляции. Однако использование только статических переменных может вызвать трудности при составлении эффективной машинной программы. Во многих случаях заранее неизвестен размер той или иной структуры данных, или структура может изменяться в процессе выполнения программы. Одна из подобных структур - последовательный файл - уже ранее рассматривалась.

В языке Паскаль предусмотрена возможность использования **динамических** величин. Для них выделение и очистка памяти происходит не на этапе трансляции, а в ходе выполнения самой программы. Для работы с динамическими величинами в Паскале предусмотрен специальный тип значений - *ссылочный*. Этот тип не относится ни к простым, ни к составным. Переменные ссылочного типа, или *указатели*, являются статическими переменными. Значением переменной ссылочного типа является адрес ячейки - места в памяти соответствующей динамической величины. Свое значение ссылочная переменная получает в процессе выполнения программы, в момент появления соответствующей динамической величины.

Переменные ссылочного типа (указатели) вводятся в употребление обычным путем с помощью их описания в разделе переменных, а их тип, указывающий на тип создаваемых в программе соответствующих динамических величин, тоже определяется либо путем задания типа в описании переменных, либо путем указания имени ранее описанного типа.

Значением указателя является адрес ячейки, начиная с которой будет размещена в памяти соответствующая динамическая величина. Задание ссылочного типа выполняется по схеме

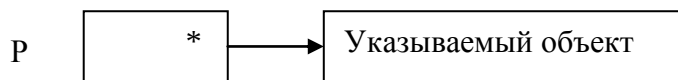
type <имя ссылочного типа> = [^] <имя типа динамической величины>
(значок [^] указывает на то, что величина является динамической).

Например:

```
type
    p = ^ integer;
    q = ^ record
        x: integer;
        y: string [20]
    end;
```

Объявлены два ссылочных типа - p и q. Первый - указатель на целочисленные значения, второй - на двухполевую запись.

Связь указателя с динамическим объектом можно изобразить следующим образом:



На этой схеме p - имя указателя; звездочкой изображено значение указателя, а стрелка отражает тот факт, что значением указателя является адрес объекта (ссылка на объект), посредством которого объект и доступен в программе.

В некоторых случаях возникает необходимость в качестве значения указателя принять «пустую» ссылку, которая не связывает с указателем никакого объекта. Такое значение в Паскале задается служебным словом nil и принадлежит любому ссылочному типу. Результаты выполнения оператора p:=nil можно изобразить

следующим образом:



Имея объявленные типы, можно обычным образом описывать переменные этих типов.

Например,

```
var i: p; zap: q;
```

Динамические переменные базовых типов можно вводить прямо в разделе «описания пере-

менных»:

var i: ^integer;

Описание указателя еще не резервирует память для значения соответствующего динамического объекта. Так, каждое вышеприведенное описание выделяет в памяти два байта для записи адреса * - значения указателя, но никакой величины типа p или q в памяти не образуется и даже адреса * еще нет.

Для порождения динамического объекта, на который указывает ссылочная переменная i, сложит стандартная процедура new(i), где new - «новый» - имя процедуры, i - имя указателя, я

Процедура new(i) выполняет две функции:

1) резервирует место в памяти для размещения динамического объекта соответствующего типа с именем i;

2) указателю i присваивает адрес динамического объекта i.

Однако, узнать адрес динамической переменной с помощью процедуры writeln(i) нельзя.

Динамические объекты размещаются по типу стека в специальной области памяти — так называемой «куче», свободной от программ и статических переменных. Как уже было отмечено, символ ^ после имени указателя означает, что речь идет не о значении ссылочной переменной, а о значении того динамического объекта, на который указывает эта ссылочная переменная. Так, если в программе имеется описание переменной

var: ^integer

то при выполнении процедуры new(i) порождается динамическая переменная типа integer. Если затем будет выполнен оператор присваивания

i:= 58

то этой динамической переменной будет присвоено значение 58.

Имя ссылочной переменной с последующим символом ^ называют «переменной с указателем». Именно она синтаксически выполняет роль динамической переменной и может быть использована в любых конструкциях языка, где допустимо использование переменных того типа, что и тип динамической переменной.

Так, если j - статическая переменная типа integer, то допустимы операторы присваивания

j:=j+i^2; i:=i^div 3+4; i:=sqrt(i^) и т.д.

Если ссылочная переменная b указывает на массив

type mas = array[1... 100] of integer, ,

var b: ^mas

то в этом случае переменные с указателем могут выглядеть, например, так:

b^[2], b^[k+5].

Если в процессе выполнения программы некоторый динамический объект p^, созданный в результате выполнения оператора new(p), становится ненужным, то его можно уничтожить (очистить выделенное ему место в памяти) с помощью стандартной процедуры dispose(p). В результате выполнения оператора вида dispose(p) динамический объект, на который указывает ссылочная переменная p, прекращает свое существование, занимаемое им место в памяти становится свободным, а значение указателя p становится неопределенным (но не равным nil).

Если до вызова процедуры dispose(p) имел пустое значение nil, то это приведет к «зависанию» программы.

Если же до вызова этой процедуры указатель p не был определен, то это может привести к выходу из строя операционной системы.

Значение одного указателя можно присвоить другому указателю того же типа. Можно также указатели одинакового типа сравнивать друг с другом, используя отношения «=» или «<>».

Стандартные процедуры new и dispose позволяют динамически порождать программные объекты и уничтожать их, что дает возможность использовать память машины более эффективно.

Связанные списки данных. Несмотря на богатый набор типов данных в Паскале, он не исчерпывает всего практически необходимого для разработки многих классов программ. В частности, из разнообразных *связанных структур данных* в языке стандартизированы массивы и файлы, а кроме них могут потребоваться и схожие с ними, но иные структуры. Для них характерны, в частности, следующие признаки:

а) неопределенное заранее число элементов;

б) необходимость хранения в оперативной памяти.

Средство для реализации таких структур дает аппарат динамических переменных.

Простейшей из обсуждаемых структур является однонаправленный список. Он строится подобно очереди на прием к врачу: пациенты сидят на любых свободных местах, но каждый из них знает, за кем он в очереди (т.е. данные размещаются на свободных местах в памяти, но каждый элемент содержит ссылку на предыдущий или следующий элемент). Поскольку количество пациентов заранее не очевидно, структура является динамической.

Другая подобная структура, с упоминанием о которой мы уже не раз встречались - **стек**. Его моделью может служить трубка с запаянным концом, в которую вкатывают шарики. При этом реализуется принцип «последним вошел - первым вышел». Возможное количество элементов в стеке не фиксировано.

Остановимся на примере стека и покажем его программную реализацию. Технически при этом следует решить ряд задач, из которых наиболее специфическими являются

- а) связывание последующих компонентов стека;
- б) смещение ссылок при каждом движении по стеку.

Из-за необходимости хранить не только значение каждого элемента, но и соответствующую ссылку на последующий элемент, каждый из элементов будем хранить в виде двухполевой записи, в которой первое поле - значение элемента, а второе - ссылка на следующий элемент. Схематически эту структуру можно описать следующим образом

→ a → a ... → a → a nil

(элементу, который пришел первым, ссылаться не на что, о чем свидетельствует «пустая ссылка» nil).

Пусть для конкретности элементы стека - действительные числа, и при последовательном заполнении состояния стека будут

1,75		
35,7	1,75	
-6,94	35,7	1,75

Приведенная ниже программа включает две процедуры: добавления очередной компоненты к стеку и изъятия ее. Программа формирует указанный стек и производит контрольное извлечение из него. Каждое действие снабжено комментарием, который поможет разобраться в программе. Следует обратить особое внимание на организацию смещения ссылок.

Программа 28

```
program stack; (формирование динамической структуры - стека)
type s^StackComp; StackComp=record
b:real;
p;s;
end;
var a:s; k:real;

procedure Dobavl(k:real); (процедура добавления к стеку числа «к»)
var j : s ;
begin new(j); (создание новой динамической записи)
j^.b:=k; (внесение следующей компоненты стека)
j^.p:=a; (смещение указателя) a:=j (формирование новой записи)
end;
procedure Vzjat(var k:real); (процедура чтения из стека)
var j :s;
begin j:=a;
k:j^.b; (k-значение последней компоненты)
a:=j^.p; (перенастройка указателя)
dispose(j) (ликвидация использованной записи)
end;
begin (пример записи и чтения)
a:=nil; Dobavl(1.75) ;
```

```

Dobavl(35.7);
Dobavl(-6.94);           (    сформирован стек: -6.94, 35.7, 1.75)
Vzjat(k); writeln(k);    (контрольное взятие из стека)
Vzjat(k); writeln(k);    (еще одно взятие из стека)
Vzjat(k); writeln(k)      (еще одно взятие из стека)
end.                      (результат работы программы: -6.94 35.7 1.75}

```

Контрольные вопросы

1. В чем принципиальное различие статических и динамических переменных?
2. 2 Как реализуется связь указателя с динамическим объектом?
3. 3. Что такое «связанный список данных»?

3.7. РАБОТА С ГРАФИКОЙ

Машинная (компьютерная) графика - одно из важных направлений в современной прикладной информатике. В отличие от базового Паскаля, современные версии содержат мощные средства разработки графических программ. Рассмотрим часть соответствующих возможностей Турбо-Паскаля, в котором они реализованы с помощью стандартного модуля Graph.

Модуль представляет собой мощную библиотеку графических подпрограмм универсального назначения, рассчитанную на работу с наиболее распространенными графическими адаптерами CGA, EGA, VGA, SVGA IBM-совместимых персональных компьютеров.

Подключение модуля Graph, три к программе выполняется директивой
uses graph;

Инициализация графического экрана осуществляется с помощью процедуры Initgraph. Драйвер поддерживает тот или иной режим экрана, табл. 3.1.

Таблица 3.1

Некоторые сведения о драйверах и определяемых ими режимах

Адаптер	Драйвер	Режим (Номер, имя)	Разрешимость	Число страниц
EGA	EGA	OEgalo	640x200	4
		1 Egahi	640x350	2
VGA	VGA	OVgalo	640x200	4
		IVgalo	640x350	2
		2Vgalo	640x480	1

Процедура инициализации в Турбо-Паскале имеет три аргумента:

Initgraph(<драйвер>, <режим>, '<путь к драйверу>').

Она может быть выполнена так:

```

uses graph;
var gd, gm: integer; {переменные gd и gm определяют драйвер и режим}
begin
  gd:=vga; gm:=vgahi;
  initgraph(gd,gm.'d:\tp551);

```

Первые две команды можно заменить одной:

```
gd:=detect
```

Целая константа detect=0 в модуле Graph автоматически распознает драйвер и устанавливает режим максимального разрешения для данной машины.

Процедура closegraph освобождает память от драйвера и устанавливает режим работы экрана, который был до инициализации графики.

Для обнаружения ошибок в графике применяются функции `graphresult` и `grapherrormsg` (код ошибки). Последняя выдает строку сообщения о характере ошибки, соответствующей коду. Инициализация графического режима с проверкой ошибок может быть выполнена в программе следующим образом:

```

uses graph; var gd, gm, errorcod: integer;
begin
    gd:=detect; initgraph(gd,gm,"");
    errorcod:=graphresult;
    if errorcod <> grok then
        begin
            writeln('ошибка графики');
            writeln(grapherrormsg(errorcod));
            halt
        end;

```

Процедура `Halt` останавливает выполнение программы и возвращает управление операционной системе.

Для формирования палитры используется система смешения красного, зеленого и синего цветов и изменения яркости луча. Цвет задается номером из списка цветов палитры в интервале 0 .. 15.

Процедуры `setcolor(<цвет>)` и `setbkcolor(<цвет>)` устанавливают текущий цвет рисунка и цвет фона. При инициализации графики по умолчанию устанавливается черный фон и белый цвет рисунка.

В табл. 3.2 указаны основные процедуры для модуля `Graph`, применяющиеся для построения простейших геометрических примитивов.

Координаты точек воспринимаются в «экранной» системе координат, в которой начало - верхний левый угол экрана, ось «х» направлена вниз, ось «у» -направо. Максимальные значения координат определяются разрешимостью экрана (см. табл. 3.1).

Первый аргумент процедуры `setlinestyle(a,b,t)` `a` - стиль линии второй параметр `b` -«образец» - имеет значение 4, если `a=4`, в остальных случаях `b=0`; третий параметр `t` -толщина линии - может иметь значение 1 (нормальная толщина) или 3 (жирная линия).

Таблица 3.2

Основные процедуры модуля `Graph`

Заголовок процедуры	Геометрический смысл
<code>putpixel(x,y,c)</code>	Построить точку (x,y) цветом c
<code>setlinestyle(a,b,t)</code>	Установить стиль, образцов и толщину линий
<code>line(x1,y1,x2,y2)</code>	Соединить две точки отрезком
<code>rectangle(x1,y1,x2,y2)</code>	Построить прямоугольник с заданными концами диагонали и сторонами, параллельными осям координат
<code>circle(x,y,r)</code>	Построить окружность с центром (x,y) и радиусом r
<code>arc(x,y,a,b,r)</code>	Построить дугу окружности: a,b - начальный и конечный угол в Градусах
<code>ellipse(x,y,a,b,rx,ry)</code>	Построить эллиптическую дугу: rx, ry - полуоси эллипса
<code>setfillstyle(t,c)</code>	Установить стиль заливки и ее цвет
<code>fillellipse(x,y,rx,ry)</code>	Построить закрашенный эллипс, используя цвет рисунка
<code>floodfill(x,y,cg)</code>	Закрасить фигуру до границы с цветом cg; (x,y) - внутренняя точка фигуры
<code>bar(x1,y1,x2,y2)</code>	Построить столбец, используя тип и цвет заливки
<code>pieslice(x,y,a,b,r)</code>	Построить и закрасить сектор круга

sector(x,y,a,b,rx,ry)	Построить и закрасить эллиптический сектор
settextstyle(f,n,d)	Установить шрифт, направление вывода и размер символа текста '
outtextxy(x,y,st)	Вывести строку st, начиная с точки (x,y)
outtext(st)	Вывести строку, начиная с точки расположения текущего указателя

Первый аргумент процедуры setfillstyle(t,c) - тип заливки t - принимает значения из интервала 0..12. Наиболее употребителен тип t = 1 - заполнение фигуры текущим цветом.

Для вывода текста на графический экран сначала выполняется процедура settextstyle(f,n,d), устанавливающая шрифт f, направление вывода n и размер символов (параметр d). При f = 0 используется стандартный точечный шрифт, встроенный в систему Турбо-Паскаль. С использованием других шрифтов познакомимся ниже. Направление вывода n принимает значения 0 (горизонтальный вывод) и 1 (вертикальный вывод).

Размер букв определяется параметром d, принимающим значения из интервала 1..10. Если d = 1 и f = 0, то каждый символ занимает квадрат 8*8 точек, при d > 1 сторона квадрата умножается на d.

Далее, с помощью процедуры outtextxy(x,y,st) строка st выводится на экран, начиная с точки (x,y).

Например:

```
settextstyle(0,0,2); outtextxy(100,200,'горизонтальная строка');
outtextxy(100,230,'размер увеличен вдвое');
```

Примеры графических программ

Пример 1: программа рисует звездное небо с 400 «звездами», вспыхивающими постепенно, и полную желтую луну.

Программа 29

```
program sky;
uses crt, graph;
var k, gd, gm: integer;
begin
  gd:=detect;
  initgraph(gd, gm, ' '); randomize;
  for k:=1 to 400 do
    begin putpixel(random(640), random(480), random(15)+1); delay(10);
    end;
  setfillstyle(1, 14); setcolor(14);
  circle(550, 80, 30); floodfill(550, 80, 14);
  repeat until keypressed; closegraph
end.
```

Пример 2. Узор. Используя простейшие геометрические образы строят замечательные графические изображения. Ниже приведена программа изображения муарового узора, полученного пересечением двух семейств расходящихся отрезков прямых. Качество и изображение получаемого узора зависит в основном от трех параметров: k1, k2 - расстояний между отрезками слева и справа; h - смещения вниз (вверх) всего семейства.

Программа 30

```
program uzor;
uses crt, graph;
var gd, gm, errCode, i, k1, k2, h; integer;
begin
  k1:=8; k2:=3; h:=110; gd:=Detect; InitGraph(gd, gm, ' ');
```

```

errCode:=GraphResult ;
if errCode = grOk then begin
    setcolor(green); *
    for i:=1 to (420 div kl) do
        begin line(0,i*kl,640,i*k2+h); line(0,i*k2+h,640,i*kl);
            end;
        repeat until keypressed; CloseGraph;
    end else writeln('errCode=',errCode)
end.

```

Процедуры построения прямоугольных фигур удобно использовать, в частности, при построении схем, диаграмм.

Пример 3: программа 31 строит столбчатую диаграмму, наглядно отражающую числовую информацию о населении 6 крупных городов мира: Токио, Гамбурга, Москвы, Бангкока, Мехико и Парижа.

Программа 31

```

program colon;
uses crt,graph;
const m:array[1..6] of real=(11500,2300,9700.5100,12400,8200) ;
      name:array[1..6] of
      string=('Токио','Гамбург','Москва','Бангкок','Мехико','Париж');.
      var gd,gm,k,n,s:integer; st:string[6];
begin
gd:=detect;
initgraph(gd,gm,' ');
setcolor(15);
setlinestyle(0,0,1);
line(60,400,620,400);
line(60,400,60,100) ;
settextstyle(0,0,1);
    for k:=1 to 12 do begin n:=1000*k; str(n,st);
        outtextxy(10,400-20*k-4,st);
        line(60,400-k*20,65,400-k*20)
        end;
        setcolor(14); settextstyle(0,0,2);
        outtextxy(120,20, 'Население городов (тысяч)');
settextstyle(0,0,1);
        for k:=1 to 6 do begin setfillstyle(1,k+2);
            bar(100+(k-1)*70,400,100+k*70,round(400-m[k]/1000*20)) ;
            outtextxy(100+(k-1)*70+4,450,name[k]) ;
        end;
        repeat until keypressed;
        closegraph
end.

```

Процедура bar3d(x1,y1,x2,y2,d,top) рисует трехмерный столбец, глубина которого определяется параметром d. Последний параметр процедуры top - логического типа. Если top=true, рисуется верхнее основание столбика, в противном случае оно не изображается, что позволяет рисовать столбики один над другим. Диаграммы с трехмерными столбиками красивы, но их труднее создавать.

Построение графиков функций. Построение графиков функции - неотъемлемая часть большинства программ, предназначенных для обучения математике, физике. Ниже представлен пример программы построения графика функции $y=x*x*\sin(1/x)$. на произвольном отрезке $[a, b]$. Количество точек графика (параметр n) также задается произвольно (точка $x=0$ исключается, так как в ней функция не определена). В программе также определяются величины $t1=(x1-x0)/(b-a)$ и $t2=(y1-y0)/(2m)$, которые означают масштабы по осям X и Y соответственно.

График рассматриваемой функции представлен на двух отрезках $[a, b]$ и $[-0.1, 0.1]$. Чтобы построить график другой функции, достаточно задать ее аналитический вид в описании функции (function f).

Программа 32

```

program grafik;
uses crt, graph;
var gd, gm, errCode : integer; a, b : real; n : integer;
function f(x:real):real;
begin if x<>0 then f:=x*x*sin(1/x);
end;
procedure grafun(x0,x1,y0,y1,n :word;a,b:real);
var h,m,x, tl,t2 : real; i, u,v,xv,yv : word;
begin
    h:=(b-a)/n; (поиск максимума f(x)) m:=abs(f(a));
    for i:=1 to n do if m<abs(f(a+i*h)) then m:=abs(f(a+i*h)) ;
    tl:=(x1-x0)/(b-a);
    t2:=(y1-y0)/(2*m); (построение координатных осей)
    setfillstyle(1,15); bar(x0-5,y0-5,x1+5,y1+5) ;
    xv:=round(x0-a*tl); yv:=round((y0+y1)/2) ;
    setcolor(1); line(xv,y0,xv,y1);
    line(x0,yv,x1,yv) ;
    Moveto(x0,yv-round(f(a)*t2)); (установка курсора в начало гра-
фика)
    setcolor(3); (построение графика) for i:=1 to n do begin
    x:=a+i*h; u:=x0+round((x-a)*tl);
    v:=yv-round(f(x)*t2); lineto(u,v);
    end;
end; (конец процедуры) begin
    clrscr; write ('введи a,b и n : '); readln(a,b,n); gd^Detect;
    InitGraph (gd,gm, ' ' ) ;
    errCode:=GraphResult ;
    if errCode = grOk then begin
        grafun(100,500,50,300,n,a,b);
        grafun(550,620,10,100,200,-0.1,0.1);
        repeat until keypressed; CloseGraph;
    end else writeln( "ezzCode=' ',errCode)
end.

```

Для изображения поверхностей, определяемых функцией двух переменных $z=f(x,y)$, можно использовать разные способы. Одним из них является метод построения семейства одномерных графиков функции $z=f(x,y)$ от одной переменной x или y различных фиксированных значениях другой. Это может быть хорошей тренировкой для самостоятельной работы по освоению графики.

Построение движущихся изображений. Особую ценность в графике представляет организация движения фрагментов рисунка. Наиболее просто это сделать по следующему плану:

- нарисовать фрагмент в нужном месте экрана;
- стереть фрагмент, рисуя его цветом фона или используя процедуру cleardevice;
- снова нарисовать фрагмент в другом месте экрана.

Такой способ осуществлен в программе billiard, где два шарика радиусом 5 пикселей разных цветов двигаются с одинаковой скоростью внутри зеленого прямоугольника, построенного с помощью процедуры bar.

Процедура blou измеряет смещение центра шарика от сторон прямоугольника по каждой оси и, если это смещение на следующем шаге цикла станет меньше радиуса, изменяет направление движения, моделируя поведение упругого шара при ударе о стенку.

В программе также рассмотрена ситуация соприкосновения шариков во время их движения. Она решается примитивно просто: каждый шарик меняет направление своего движения на проти-

Программа 33

```

program billiard;
uses crt,graph;
var x,y,dx,dy,gd,gm:integer; xl,yl,dxl,dyl:integer;
procedure blow(a,b:integer; var c,d:integer);
begin if (a<107) or (a>523) then c:=-c;
      if (b<107) or (b>363) then d:=-d;
end;
begin
gd:=detect; initgraph(gd,gm,''); setcolor(14);
setlinestyle(0,0,1); rectangle(99,99,531,371) ;
setfillstyle(1,3); bar(100,100,530,370) ;
x:=320; y:=240; dx:=2; dy:=2;
xl:=320; yl:=200; dxl:=-2; dyl:=-2;
repeat circle(x,y,5); setcolor(4); circle(xl,yl,5);
blow(x,y,dx,dy); blow(xl,yl,dxl,dyl); delay(10);
if (abs(x-xl)<=10) and (abs(y-yl)<=10) then begin
    dx:=-dx; dy:=-dy; dxl:=-dxl; dyl:=-dyl; delay(300)
end;
setcolor(3); circle(x,y,5);
x:=x+dx; y:=y+dy;
setcolor(3); circle(xl,yl,5) ;
xl:=xl+dxl; yl:=yl+dyl; setcolor(14)
until keypressed;
closegraph
end.

```

Еще один способ организации движения на экране, широко применяющийся в компьютерных играх, связан с использованием нескольких экранных страниц. В режиме Vgamed их две, а в режиме Vgalo - четыре. Страницы имеют номера: 0,1,... В любой момент времени одну из страниц можно сделать видимой и посмотреть ее содержимое на экране с помощью процедуры `setvisualpage(номер)`. Визуальная страница обычно пассивна, т.е. на ней нельзя выполнять графические процедуры. Другую страницу можно объявить активной с помощью процедуры `setactivepage(номер)`. Активная страница невидима для пользователя. На ней можно подготовить другой рисунок. В следующий момент можно поменять роли страниц, т.е. визуальную сделать активной и невидимой и на ней рисовать следующий кадр, а бывшую активную сделать визуальной и показать объект в новом месте экрана.

В программе `helicopter` с помощью страниц моделируется вращение винтов вертолета. Две процедуры `vert1` и `vert2` подготавливают рисунки вертолета с разным положением винтов. Изображения выводятся на разные страницы, которые потом по очереди становятся то видимыми, то активными.

В программе организовано также движение вертолета по эллиптической орбите. Центр орбиты - точка (x_0, y_0) - располагается в центре экрана, числа $a=250$ и $b=130$ - горизонтальная и вертикальная полуоси эллипса. Положение вертолета на орбите 'вычисляется по формулам

$$u=x_0+\text{round}(a*\cos p), t=y_0+\text{round}(b*\sin p),$$

где p - текущий угол, образованный радиус-вектором точки эллипса с осью абсцисс.

Программа 34

```

program helicopter;
uses crt,graph;
const step=0.01;
var gd,gm,u,t,z,k,x0,y0rc,ac,vi:integer; p,a,b:real;
procedure vert1(x,y:integer) ;
begin

```



```

cleardevice; setcolor(14);
setlinestyle (0,0,1); setfillstyle(1,3) ;
line(x+12,y,x+36,y);line(x+24,y,x+24,y+8) ;
fillellipse(x+24,y+14,12,7); moveto(x+18,y+20);
lineto(x,y+20); lineto(x,y+14);
line(x+18,y+28,x+30,y+28); line(x+24,y+27,x+24,y+21) ;
end;
procedure vert2(x,y:integer);
begin
  cleardevice; setcolor(14);
  setlinestyle(0,0,1); setfillstyle(1,3) ;
  line(x+22,y,x+26,y); line(x+24,y,x+24,y+8) ;
  fillellipse(x+24,y+14,12,7); moveto(x+18,y+20) ;
  lineto(x,y+20); lineto(x,y+14) ;
  line(x-3,y+14,x+3,y+14); line(x+24,y+27,x+24,y+21) ;
  line(x+18,y+28,x+30,y+28);
end;
begin
  gd:=vga; gm:=vgamed; initgraph(gd,gm,' ');
  x0:=getmaxx div 2; y0:=getmaxy div 2;
  z:=1; p:=pi; a:=250; b:=130; ac:=0; vi:=1;
  for k:=1 to 800 do
    begin u:=x0+round(a*cos(p)); t:=y0-round(b*sin(p));
    • setactivepage(ac); setvisualpage(vi);
      if z=1 then vert1(u,t) else vert2(u,t); z:=-z;
      delay(50); p:=p+step; c:=ac; ac:=vi; vi:=c;
    end; closegraph
  end.

```

Контрольные вопросы и задания

1. Какие бывают режимы графического экрана?
2. Охарактеризуйте возможности процедур модуля Graph.
3. Какие есть способы построения движущихся изображений?
4. Постройте трехмерные столбчатую и круговую *анаграммы* для примеров, рассмотренных в данном параграфе.
5. Изобразите поверхность функции $z=\sin(x)+\cos(y)$, предусмотрев удаление «невидимых линий».
6. Создайте демонстрационную модель идеального газа в замкнутом объеме.

3.8. ТУРБО-ОБОЛОЧКИ. ВЕРСИИ ПАСКАЛЯ

Система программирования Турбо-Паскаль 3.0. После запуска программы turbo.exe на экране появится меню системы

Logged drive : A									
Work file:									
Main file:									
Edit	Compile	Run	Save	eXecute	Dir	Quit	compiler	Options	
Text:	0 bytes (7BF5-7BF5)								
Free:	19472 bytes (7BF6-C806) >								

Рис. 3.2. Главное меню Турбо-Паскаля 3.0

Соответствующие команды выполняются при нажатии выделенной большой буквы.
 Logged drive: команда «L» осуществляет выбор дисководов: на запрос

(выбор устройства)	NEW DRIVE ввести одну из букв от А до Р, обозначающую номер дискового;
Work file: (рабочий файл)	команда «W» осуществляет выбор рабочего файла, на запрос Work file name ввести имя вашего файла;
Main file: (основной файл)	команда «M» выбирает основной файл для программ, использующих директивы \$I: аналогична команде «W»;
Edit (редактор)	команда «E» вызывает текстовый редактор; если не выбран файл, то будет запрос; инструкцию по работе с редактором см. ниже;
Compile (компиляция)	команда «C» осуществляет компиляцию программы, компиляция проводится в память, режим компиляции (в COM или CHN файлы) меняют по команде «O» (см. compiler Options);
Run (запуск)	команда «R» запускает откомпилированную или неоткомпилированную программу;
Save (сохранить)	команда «S» осуществляет запись рабочего (сохранить) файла с сохранением предыдущей версии с расширением .BAK;
EExecute (выполнять)	команда «X» осуществляет вызов и выполнение любого COM-файла;
Dir (оглавление файлов);	команда «D» осуществляет просмотр директории
Quit (прервать)	команда «Q» осуществляет выход из системы; если файл не записан, то будет запрос;
compiler Options (опции компилятора)	команда «O» выбирает режим компиляции с помощью меню



(команды «M», «C» и «H» осуществляют выбор режима компиляции в память, в com-файл и в chn-файл (откомпилированный файл без библиотеки), команда «Q» возвращает в основное меню)

Описание языка Паскаль, проведенное в §3, соответствует версии 3. Исключение составляет раздел работы с графикой, в котором использован универсальный модуль Graph для версий 5 и выше. Таким образом, практически все приведенные в главе примеры могут быть реализованы в среде Турбо-Паскаль 3 (и, разумеется, в более поздних версиях)

Система программирования Турбо-Паскаль 5.5. Ниже перечисляются основные расширения возможностей версии Турбо-Паскаля 5.5:

- допускается изменение структуры Паскаль-программы, заголовок программы (program) не обязателен, разделы глобальных описаний могут следовать в любом порядке;
- расширен синтаксис описания констант, разрешено определять типизированные констан-

ты и производить их инициализацию;

- введены шестнадцатиричные константы;
- расширен список арифметических функций;
- введены типы данных Word, Byte,
- значительно расширен список процедур и функций для работы с файлами и модулями.

Начиная с версии 4.0 в Турбо-Паскале введено понятие модуля (Unit), определены его составляющие части, а также разрешена раздельная компиляция модулей. Имеются стандартные модули Crt, Svsstem, DOS, Graph, Turbo3. Graph3. Последние модули призваны осуществлять совместимость с предыдущими версиями.

В системе программирования Турбо-Паскаль версии 5.5 создана интегрированная среда разработчика, т е ряд специализированных средств и возможностей, объединенных оконным турбо-меню (контекстное меню). Главное меню интегрированной среды содержит следующие пункты:

File Edit Run Compile Options Debug Break/watch

Система вступает в диалог с пользователем и предоставляет возможность обратиться к контекстной помощи в любой момент.

Как правило, работа в среде начинается с загрузки текста программы или с его формирования в окне редактирования. После редактирования программы осуществляется компиляция и выполнение, которые выполняются при нажатии клавиш Ctrl+F9 (эквивалентно выбору команды run из меню Run).

Во время компиляции программы на экране появляется специальное окно, в котором формируется процесс компиляции. В случае обнаружения ошибки компиляция прерывается и активизируется встроенный редактор, в котором курсор указывает место ошибки. Клавиша F1 в этот момент выводит информацию из службы помощи по диагностике ошибки. Исправив ошибку, можно опять отправить программу на компиляцию. После удачной компиляции программа выполняется и результаты ее работы можно просмотреть нажатием клавиш Alt+F5. В процессе компиляции выявляются лишь синтаксические ошибки. Правильность работы программы необходимо проверять специальными приемами. В интегрированной среде разработчика предусмотрен отладчик, позволяющий осуществлять трассировку программы или ее отдельных участков, прерывать выполнение в заданных точках, следить за изменениями значений различных переменных. Трассировкой называют пошаговое исполнение программы, при котором за один шаг выполняются все операторы одной строки с последующим ожиданием. Очевидно, что трассировка эффективнее работает для исходного текста, в котором в строке помещают не более одного оператора. При этом появляется возможность анализировать результаты работы каждого оператора в отдельности.

Сеанс отладки обычно начинается с команды Trace into (F*) или Step over (F8) из меню Run. Выборочная трассировка отдельных участков исходного текста осуществляется с помощью команд Toggle breakpoint из меню Break/Watch и Go to cursor меню Run, Для наблюдения за значениями переменных и выражений существует окно просмотра Watch (F6) в меню Debug. В нем отображаются текущие значения выбранного выражения. Сеанс отладки программы необходимо завершать командой Program reset (Ctrl+F2) из меню Run.

Система программирования Турбо-Паскаль 6.0. Главное меню интегрированной среды содержит следующие пункты. Е - системное меню;

File - работа с файлами (сохранение, загрузка, связь с операционной системой);

Edit -редактирование текущего файла (стандартные возможности встроенного текстового редактора);

Search - поиск и замена фрагментов текста;

Run -запуск программы на выполнение;

Compile -компиляция программы;

Options - установка опций интегрированной среды;

Debug - установка параметров отладки программы;

Window - работа с окнами;

Help - система помощи и подсказок.

Версия 6.0 является развитием и усовершенствованием версии 5.5. Существенно перерабо-

тана интегрированная среда пользователя. В ней появилась возможность работать в многооконном режиме, редактируя несколько файлов одновременно. Допускается работа с «мышью». Существенно улучшен сервис для работы с окнами и с системой помощи Help, которая контекстно вызывается из любого окна, а в окне Help допустимы некоторые команды редактирования. Увеличился максимальный размер редактируемого файла и имеется возможность менять цветовую палитру самой интегрированной среды.

Еще одна возможность, появившаяся в Турбо-Паскале 6.0 - объектно-ориентированное проектирование программ. Подобная технология стала возможной благодаря новому типу данных «объект». Объект - это структура данных, содержащая поля данных (подобно записи) различных типов и заголовки методов. Метод - это процедура или функция, объявленные внутри объявления элемента типа «объект». Большую популярность среди программистов приобрела объектно-ориентированная библиотека TurboVision, включающая объекты, управляющие

- перекрывающими окнами с изменяемыми размерами;
- выпадающими меню;
- диалоговыми окнами;
- работой мыши;
- кнопками, полосами скроллинга.

Следует отметить, что сама интегрированная среда Турбо-Паскаля 6.0 разработана с использованием библиотеки TurboVision.

Система программирования Турбо-Паскаль 7.0. Главное меню интегрированной среды Турбо-Паскаль 7.0 в дополнение к меню Турбо-Паскаля 6.0 содержит

Tools - инструментальные программные средства.

По сравнению с предыдущей версией в интегрированной среде появилась возможность настройки компилятора на работу в трех режимах: обычном режиме MS DOS (Real), защищенном режиме (Protected) и в режиме операционной среды Windows (Windows). Введена новая команда Object Browser - просмотр модулей, объектов и символов при редактировании исходного текста. Появилось удобное средство - синтаксическая подсветка, которая обеспечивает подсвечивание управляющих структур, зарезервированных слов, идентификаторов, строк и т.п. Новая секция меню Tools предназначена для передачи управления внешним программам и создания собственных инструментальных программных средств. Нажатие клавиш Alt+F1 (или правый щелчок мыши) активизирует локальное меню, чувствительные к контексту (Browse, Edit, Help, Message, Watch).

В версии 7.0 расширена библиотека стандартных модулей. Помимо известных в версии 6.0 модулей Crt, Graph, Graph3, Overlay, String, System, Turbo3 появились WinAPI, WinCrt, WinDos, WinPrn, WinTypes, WinProcs.

Контрольные вопросы

1. В чем состоят основные различия версий Турбо-Паскаля?
2. Какие возможности версии Турбо-7.0 отсутствуют в предыдущих версиях?

3.9. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЮ ТУРБО-ПАСКАЛЯ

СТАНДАРТНЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ

Принятые обозначения:

type -любой тип;
string - любой строковый тип;
file -любой файловый тип;
scalar - любой простой тип;
pointer -любой ссылочный тип.

Замечание: если в списке параметров не указан спецификатор типа, то в качестве данного параметра процедура допускает параметр любого типа; в некоторых процедурах допускаются несколько вариантов заголовков процедур.

ПРОЦЕДУРЫ ВВОДА-ВЫВОДА

Процедура **Read**

```
Read(var F:file of type; var v:type);  
Read(var F:text; var I:intnger);  
Read(var F:text; var R:real);  
Read(var F:text; var C:char);  
Read(var F:text; var S:string);
```

где F - имя переменной файлового типа, а var I,R,C,S - одна или несколько переменных, совпадающих по типу с типом элементов файла.

Процедура **Readln**

```
Readln(var F:text; var var1,var2,...varN);  
Readln(var var1 ,var2,...varN);  
Readln(var F:text);  
Readln;
```

где F - имя переменной типа text, а var1...varN - переменные типа Char. Если опущено имя файла, то по умолчанию принимается имя типа INPUT. Если опущен список переменных, то происходит переход на другую строку.

Процедура **Write**

```
Write(var F:file of type; var v:type);  
Write(var F:text; var I:integer);  
Write(var F:text; var B:boolean);  
Write(var F:text; var C:char);  
Write(var F:text; var S:string).
```

Процедура **Writeln**

```
Writeln(var F:text; var var1 ,var2,... varN);  
Writeln(var var var1 ,var2,... varN);  
Writeln(var F:text);  
Writeln.
```

СТАНДАРТНЫЕ ФУНКЦИИ

Функции арифметические

Abs(x); Arctan (x); Cos (x); Exp (x); Ln (x); Sin (x); Sqr (x); Sqrt (x)

имеют стандартный смысл. Функция

fruc(R:Real):Real;fruc(I:Integer):Real

возвращает значение дробной части своего аргумента.

Функция

int (R:real):real; int (I:integer):real;

возвращает значение целой части своего аргумента.

Функции скалярные

Odd(x); Pred(x); Succ(x)

имеют стандартный смысл.

Функции преобразования

Chr (x); Ord (x); Round (x); Trunc (x) имеют стандартный смысл.

СТРОКОВЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ

Процедура **Delete**(var S:String;Pos, Len: Integer) убирает из строки S Len символов, начиная с Pos. Если Len больше длины S, то ничего не происходит. Если Pos+Len больше длины строки, то удаляются все символы до конца строки.

Процедура **Insert**(S:String; var D:String;Pos:Integer) вставляет строку S в строку D перед символом с номером Pos. Если Pos больше длины D, то S и D соединяются. Если длина результата при этом больше максимальной длины D, то в D записываются только самые левые символы.

Процедура **Str**(I:Integer; var S: String); **Str**(R:Real; var S:String) преобразует I или R из числа

в строку и записывает эту строку в S, причем R и I могут записываться формально, т.е. как в процедурах Write и Writeln.

Процедура **Val**(S:String; var R:Real; var P:Integer); **Val**(S:String; var I,P:Integer) преобразует строковую переменную S в число (тип зависит от типа переменной I или R соответственно). Если в строке S ошибок нет, то P равно 0, иначе значение R (или I) не определено, а P присваивается номер первого ошибочного символа.

Функция **Concat**(S1,S2,...,Sn:string):String возвращает строку, полученную конкатенацией строк S1,...,Sn. Если длина результата больше 255, то возникает ошибка времени выполнения.

Функция **Copy**(S:String; Pos,Len:Integer):String возвращает строку, полученную из Len символов строки S, начиная с позиции Pos. Если Pos больше, чем длина строки S, то возвращается пустая строка. Если Pos+Len больше, чем длина строки S, то возвращаются только символы, принадлежащие строке S.

Функция **Length**(S:String):Integer возвращает длину строки S.

Функция **Pos**(Pattern,Source:String):Integer возвращает номер символа, начиная с которого Pattern входит в Source. Если вхождения нет, то возвращается 0.

ПРОЦЕДУРЫ И ФУНКЦИИ ДЛЯ РАБОТЫ С ФАЙЛАМИ

Процедура **Assign**(var F:File; Name:String); Name - имя файла на диске, F – имя файловой переменной. После выполнения этой процедуры Name и F отождествляются. Assign не должна применяться к файлу, находящемуся в употреблении.

Процедура **BlockRead**(var F:File; var Dest:Type; Num:Integer)

F -нетипизируемый файл, Dest -переменная,

Num - количество 128-байтных записей, которые должны быть перенесены из дискового файла в переменную.

Процедура **BlockWrite**(var F:File; var Dest:Type; Num:Integer) I Num - количество 128-байтных записей, которые должны быть перенесены из переменной в дисковый файл.

Процедура **Chain**(var F:File) активизирует CHN-файл, с которым предварительной процедурой Assign было отождествлено имя F.

Процедура **Close**(var F:File); дисковый файл, отождествленный с переменной F, закрывается и в директорию диска вносятся необходимые изменения.

Процедура **Erase**(var F:File); дисковый файл, отождествленный с переменной F, уничтожается.

Процедура **Execute**(var F:File) загружает и выполняет файл с диска, с которым предварительно процедурой Assign было отождествлено имя F.

Процедура **Rename**(var F:File; Name:String); дисковому файлу, отождествленному с переменной F, присваивается новое имя Name. Rename не должна применяться к уже открытому файлу.

Процедура **Reset**(var F:File) открывает файл F для чтения и ставит указатель на начало файла.

Процедура **Rewrite**(var F:File) открывает файл F для записи, чистит его и ставит указатель на начало файла.

Процедура **Seek**(var F:File of type; Pos:Integer) устанавливает указатель файла F на Pos элемент файла.

Процедура **Flush**(var F:File) очищает буфер дискового файла, отождествленного с переменной F. Гарантирует, что следующая операция чтения будет произведена непосредственно с диска. Должна применяться к уже закрытым файлам.

Функция **Eof**(var F:File):boolean возвращает TRUE, если найден конец файла.

Функция **Eoln**(var F:Text):boolean возвращает TRUE, если найден конец строки или конец файла F.

Функция **Filepos**(var F: File of Type):Integer; **Filepos**(var F:File):Integer возвращает текущую позицию указателя файла.

Функция **Filesize**(var F: File of Type):Integer; **Filesize**(var F:File):Integer возвращает текущее количество элементов файла.

ПРОЦЕДУРЫ И ФУНКЦИИ ДЛЯ РАБОТЫ СО ССЫЛОЧНЫМИ СТРУКТУРАМИ

Процедура **Getmem**(var P:Pointer; I:Integer) отводит под область памяти, на которую будет ссылаться переменная P, 1 байт памяти.

Процедура **Mark**(var P:Pointer) запоминает текущий указатель «кучи» в переменной P.

Процедура **New**(var P:Pointer) отводит место в памяти под переменную того типа, на который указывает P.

Процедура **Release**(var P:Pointer) ставит указатель «кучи» на значение переменной P. Переменная P должна быть предварительно обработана процедурой Mark. Уничтожаются все динамические переменные, расположенные в памяти после последнего вызова процедуры Mark.

Функция **MemAvai**:Integer возвращает размер доступной для динамических переменных памяти.

Функция **Ord**(P:Pointer):Integer возвращает адрес переменной ссылочного типа P.

Функция **Ptr**(I:Integer):Rointer возвращает ссылку на адрес памяти I.

ПРОЦЕДУРЫ ДЛЯ РАБОТЫ С ТЕКСТОВЫМ ЭКРАНОМ

Процедура **CrtExit** - выдача на экран строки сброса экрана, определенной в инсталляторе.

Процедура **Crtlnit** - выдача на экран строки инициализации экрана, определенной в инсталляторе.

Процедура **ClrEol** уничтожает символы строки от курсора до крайнего правого символа без перемещения курсора.

Процедура **ClrScr** очищает экран.

Процедура **DellLine** - удаление строки, на которой находится курсор, и сдвиг всех строк, находящихся ниже этой строки, на одну вверх.

Процедура **GotoXY** (X,Y:Integer) - позиционирование курсора по координатам **XnY**.

Процедура **InsLine** - вставка пустой строки на место курсора и сдвиг всех строк, находящихся под этой строкой на одну вниз.

ПРОЧИЕ ПРОЦЕДУРЫ И ФУНКЦИИ

Процедура **Bdos**(func,param:Integer) - вызов функции Bdos с номером func, которая заносится в регистр C, и параметром param, который заносится в регистровую пару DE.

Процедура **BIOS** (func,param:Integer) - вызов функции Bios с номером func и параметром param, который заносится в регистровую пару BC.

Процедура **Delay** (Ms:Integer) - задержка программы на Ms миллисекунд.

Процедура **FillChar**(var dest; length:Integer;data:Char); **FillChar**(var dest; length:Integer;data:Byte) - заполнение length байт памяти величиной data. Начало участка памяти - адрес переменной dest.

Процедура **Halt** - аварийный выход из программы.

Процедура **Move**(var source.dest:length:Integer) - пересылка length байт, расположенных в памяти, начиная с первого байта переменной source, в участок памяти длины length, начинающийся с первого байта переменной dest.

Процедура **Randomise**- инициализация генератора случайных чисел.

Функция **Addr**(var variable):Integer; **Addr**(<function identifier>):Integer;

Addr(<procedure identifier>):Integer возвращает адрес первого байта своего аргумента.

Функция **Bdos**(funk,param:Integer):Byte аналогична соответствующей процедуре. Возвращает значение регистра A.

Функция **BdosHL**(funk,param:Integer):Integer аналогична функции Bdos. Возвращает значение регистровой пары HL.

Функция **Bios**(funk, param :Integer):Byte аналогична соответствующей процедуре. Возвращает значение регистра A.

Функция **BiosHL**(funk,param:Integer):Integer аналогична функции Bios. Возвращает значение регистровой пары HL.

Функция **Hi**(I:Integer):Integer возвращает число, младший байт которого содержит старший байт параметра. Старший байт результата равен 0.

Функция **IOresult:Integer** работает после директивы компилятору \$1-. Значение 0 соответствует отсутствию любой ошибки ввода-вывода. В противном случае возвращается произвольное ненулевое число.

Функция **KeyPresscd:Boolean** возвращает TRUE, если нажата любая клавиша на клавиатуре, и FALSE в противном случае.

Функция **Lo(I:Integer):Integer** возвращает число, младший байт которого содержит младший байт параметра. Старший байт результата равен 0.

Функция **Random(I:Integer):Integer; Random:Real** возвращает случайное число. Для первого варианта вызова это число лежит в интервале от 0 до 1, для второго -от 0 до 1.

Функция **SizeOf(var variable):Integer; SizeOf(<type identifier>):Integer** возвращает размер (в байтах) своего аргумента.

Функция **Swap(I:Integer):Integer** возвращает число, у которого по сравнению с аргументом поменялись местами первый и второй байты.

Функция **UpCase(Ch:Char):Char** возвращает верхний регистр данного символа. Если такового нет, то возвращается аргумент.

Функция **ParamCount:Integer**; возвращает число параметров в строке вызова COM-файла.

Функция **ParamStr(n:Integer):String[14]**; возвращает n-параметр из строки вызова COM-файла.

Процедура **Exit** - выполнение этой процедуры эквивалентно передаче управления на последний END процедуры, функции или основной программы, где выполнена процедура **Exit**.

§4. МЕТОДЫ И ИСКУССТВО ПРОГРАММИРОВАНИЯ

4.1. ПРОЕКТИРОВАНИЕ ПРОГРАММ

В предыдущем разделе, посвященном языку Паскаль, приведено немало примеров программ. Однако, при анализе готовой программы чаще всего не ясно, как разработчики к ней пришли. В этом разделе рассказывается об общих моментах в технологии программирования. Конечно, при разработке небольших учебных программ не все элементы этой технологии следует отрабатывать (да это и не всегда возможно по-существу), однако само ее существование должно быть осознано.

Современный подход к проектированию программ основан на декомпозиции задачи, которая в свою очередь основана на использовании абстракций. Целью при декомпозиции является создание модулей, которые представляют собой небольшие, относительно самостоятельные программы, взаимодействующие друг с другом по хорошо определенным и простым правилам. Если эта цель достигнута, то разработка отдельных модулей может осуществляться различными людьми независимо друг от друга, при этом объединенная программа будет функционировать правильно.

Различают абстракцию через параметризацию и через спецификацию. Смысл абстракции через параметризацию в том, что одним алгоритмом можно решать задачи, отличающихся различными исходными данными, задаваемыми как параметры. Смысл абстракции через спецификацию в том, что разными алгоритмами можно получить один и тот же искомый результат. При этом описываются результаты работы программы, смысл обращения к программе становится ясным через анализ ее спецификации, а не самого текста программы.

Разработка любой программы или программной системы начинается с определения требований к ней для конкретного набора пользователей и заканчивается, "эксплуатацией системы этими пользователями.

Существуют различные подходы и технологии разработки алгоритмов и программ. Хотя программирование в значительной степени искусство, тем не менее, можно систематизировать и обобщить накопленный профессиональный опыт. По современным взглядам проектирование и разработку программ целесообразно разбить на ряд последовательных этапов:

- 1) постановка задачи;
- 2) проектирование программы;
- 3) построение модели;
- 4) разработка алгоритма;
- 5) реализация алгоритма;

6) анализ алгоритма и его сложности;

7) тестирование программы;

8) документирование.

Кратко остановимся на каждом из этих этапов.

При **постановке задачи** для крупных компьютерных программ необходимо провести следующие работы:

- выработать требования (свойства, качества и возможности), необходимые для решения проблемы или достижения цели (как правило, эта деятельность носит экспертный характер);

- разработать спецификации, включающие:

- цель программы;

- граничные условия;

- описание функций системы;

- спецификации входных и выходных данных;

- верификационные требования (установление тестовых случаев);

- тип и количество документов.

В ходе этой работы выявляются свойства, которыми должна обладать система в конечном виде (замысел), описываются функции системы, характеристики интерфейса.

Чтобы приступить к решению задачи необходимо точно ее сформулировать. В первую очередь, это означает определение исходных и выходных данных, т.е. ответы на вопросы: а) что дано; б) что нужно найти. Дальнейшая детализация постановки задачи представляет собой ответы на серию вопросов такого рода:

- как определить решение;

- каких данных не хватает и все ли они нужны;

- какие сделаны допущения и т.п.

Проектирование программы. Сначала производится проектирование архитектуры программной системы. Это предполагает первичную (общую) стадию проектирования и заканчивается декомпозицией спецификаций в структуру системы. Обычно на модульном уровне по каждому модулю разрабатывается спецификация модуля:

- имя/цель - дается имя модулю и предложение о функции модуля с формальными параметрами;

- неформальное описание - обзор действий модуля;

- ссылки - какие модули ссылаются на него и на какие модули ссылается данный модуль;

- вход/выход - формальные и фактические параметры, глобальные, локальные и связанные (общие для ряда модулей) переменные;

- примечания - полезные комментарии общего характера по модулю.

Следующим шагом является детальное проектирование. На этом этапе происходит процедурное описание программы, выбор и оценка алгоритма для реализации каждого модуля. Входной информацией для проектирования являются требования и спецификации системы.

Для проектирования программ существуют различные подходы и методы. Современный подход к проектированию основан на декомпозиции, которая, в свою очередь, основана на использовании абстракции. Целью при декомпозиции является создание модулей, которые взаимодействуют друг с другом по определенным и простым правилам. Декомпозиция используется для разбиения программы на компоненты, которые затем могут быть объединены.

Методы проектирования архитектуры делятся на две группы:

- 1) ориентированные на обработку и

- 2) ориентированные на данные.

Методы, ориентированные на обработку, включают следующие общие идеи.

а) *Модульное программирование.* Основные концепции:

- каждый модуль реализует единственную независимую функцию;

- имеет единственную точку входа/выхода;

- размер модуля минимизируется;

- каждый модуль разрабатывается независимо от других модулей;

- система в целом построена из модулей. Исходя из этих принципов каждый модуль тестируется отдельно, затем после кодирования и тестирования происходит их интеграция и тестируется вся система.

б) *Функциональная декомпозиция.*

Подобна стратегии «разделяй и управляй». Практически является декомпозицией в форме пошаговой детализации и концепции скрытия информации. Каждый модуль характеризуется субъективным решением проектировщика, связь осуществляется с помощью хорошо организованных интерфейсов.

в) *Проектирование с использованием потока данных.*

Использует поток данных как генеральную линию проектирования программы. Содержит элементы структурного проектирования сверху-вниз с пошаговой детализацией:

- экспертиза потоков данных и отображение графа потока данных;
- анализ входных, центральных и выходных преобразующих поток данных элементов;
- формирование иерархической структуры программы;
- детализация и оптимизация структуры программы.

г) *Технология структурного анализа проекта.*

Основана на структурном анализе с использованием специальных графических средств построения иерархических функциональных связей между объектами системы. Эффективна на ранних стадиях создания системы, когда диаграммы просты и читаемы.

Методы проектирования, основанные на использовании структур данных, описаны ниже.

а) *Методология Джексона.*

Здесь структура данных - ключевой элемент в построении проекта. Структура программы определяется структурой данных, подлежащих обработке. Программа представляется как механизм, с помощью которого входные данные преобразуются в выходные. В методе предусматривается:

- разработка и изображение структуры входных и выходных данных;
- изображение структуры программы путем соединения изображений этих структурных элементов;
- определение дискретных операций над структурами данных;
- построение алгоритмов обработки структур данных.

б) *Методология Уорнера.*

Подобна предыдущей, но процедура проектирования более детализирована. Используются следующие виды представления проекта:

- диаграммы организации данных (описывают входные и выходные данные);
- диаграммы логического следования (логический поток этих данных);
- список инструкций (команды, используемые в проекте);
- псевдокод (описание проекта);
- определение входных данных системы;
- организация входных данных в иерархическую структуру;
- детальное определение формата элементов входного файла;
- то же самое для выходных данных;
- спецификация программы: чтение, ветвление, вычисление, выходы, вызовы подпрограмм;
- составление диаграммы (по типу блок-схем) указывающие логическую последовательность инструкций.

в) *Метод иерархических диаграмм.*

В этом методе определяется связь между входными, выходными данными и процессом обработки с помощью иерархической декомпозиции системы (без детализации). По сути используются три элемента: вход, обработка, выход.

Алгоритм проектирования по этому методу заключается в следующих шагах:

- начать с наивысшего уровня абстракции, определив вход, выход, обработку;
- соединить каждый элемент входа и выхода с соответствующей обработкой;
- документировать каждый элемент системы, используя диаграммы;
- детализировать диаграммы, используя шаги 1 - 3.

г) *Объектно-ориентированная методология проектирования.*

Основана на концепции упрятывания информации и абстрактных типов данных. Рассматриваются данные, модули и системы в качестве объектов. Каждый объект содержит некоторую структуру данных с набором процедур, знающих как работать с этими данными. По этой методо-

логии создаются абстракции по заданной проблемной области:

- определение проблемы;
- развитие неформальной стратегии, удовлетворяющей требованиям к системе;
- формализация стратегии;
- создание объектов и их атрибутов;
- определение операций над объектами;
- установка интерфейсов;
- реализация операций.

Построение модели в большинстве случаев является непростой задачей. Чтобы приобрести опыт в моделировании, необходимо изучить как можно больше известных и удачных моделей.

При построении моделей, как правило, используют два принципа: дедуктивный (от общего к частному) и индуктивный (от частного к общему).

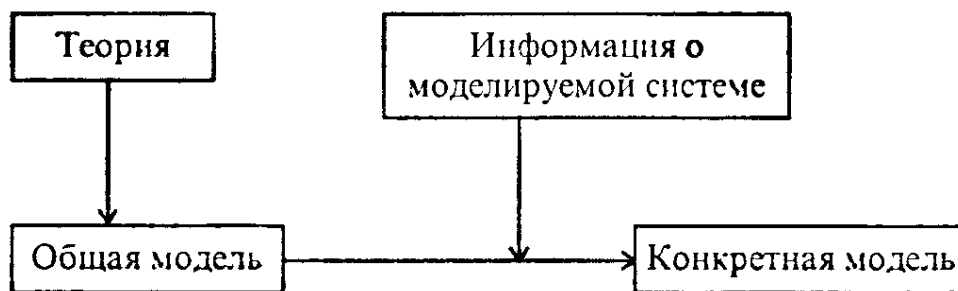


Рис. 3.3. Схема построения модели при дедуктивном способе

При дедуктивном подходе (рис.3.3) рассматривается частный случай общеизвестной фундаментальной модели. Здесь при заданных предположениях известная модель приспособляется к условиям моделируемого объекта. Например, можно построить модель свободно падающего тела на основе известного закона Ньютона $ma = mg - F_{сопр}$ и в качестве допустимого приближения принять модель равноускоренного движения для малого промежутка времени.

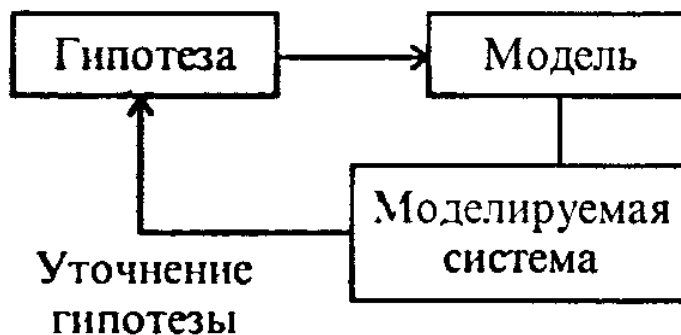


Рис. 3.4. Схема построения модели при индуктивном способе

Индуктивный способ (рис.3.4) предполагает выдвижение гипотез, декомпозицию сложного объекта, анализ, затем синтез. Здесь широко используется подобие, аналогичное моделирование, умозаключение с целью формирования каких-либо закономерностей в виде предположений о поведении системы.

Технология построения модели при индуктивном способе:

1) эмпирический этап

- умозаключение;
- интуиция;
- предположение;
- гипотеза.

2) постановка задачи для моделирования;

3) оценки; количественное и качественное описание;

4) построение модели.

Разработка алгоритма - самый сложный и трудоемкий процесс, но и самый интересный в творческом отношении. Выбор метода разработки зависит от постановки задачи, ее модели. (О не-

которых приемах и методах разработки алгоритмов говорилось ранее в гл. 1 и будет сказано в следующих разделах данной главы.) На этом этапе необходимо провести анализ правильности алгоритма, что очень непросто и трудоемко. Наиболее распространенная процедура доказательства правильности алгоритма - это прогон его на множестве различных тестов. Однако, это не гарантирует того, что не может существовать случая, в котором программа «не сработает». В общей методике доказательства правильности алгоритма предполагают, что алгоритм описан в виде последовательности шагов. Для каждого шага предлагается некое обоснование его правильности для всех подходящих входных (условиях до данного шага) и выходных данных (условиях после этого шага). Затем предлагается доказательство конечности алгоритма с окончательными исходными входными и выходными данными.

На этапе реализации алгоритма происходит конструирование и реализация алгоритма, включая:

- кодирование;
- интеграцию;
- тестирование (сертификацию).

По сути проводится перевод проекта в форму программы для конкретного компьютера, сборка системы и ее прогон при тестовых и нормальных условиях для подтверждения ее работы в соответствии со спецификациями системы. Этот этап зависит от того, какой язык программирования выбран, на каком компьютере алгоритм будет реализован. С этим связаны выбор типов данных, вводимых структур данных, связь с окружающей средой и т.п. Важно осознавать интерактивность, вид транслятора (компилятор или интерпретатор), наличие библиотек подпрограмм, модулей и объектов.

Анализ алгоритма и его сложности необходим для оценки ресурсов компьютеров, на которых он будет работать, времени обработки конкретных данных, приспособления в работе в локальных сетях и телекоммуникациях. Хотелось бы также иметь для данной задачи количественный критерий для сравнения нескольких алгоритмов с целью выбора более простого и эффективного среди них.

Перед началом эксплуатации программы необходим этап ее отладки и **тестирования**.

Тестирование - это процесс исполнения программ с целью выявления (обнаружения) ошибок. Тестирование - процесс деструктивный, поэтому считается, что тест удачный, если обнаружена ошибка. Хорошим считается тест, который имеет большую вероятность обнаружения еще не выявленной ошибки. Удачным считается тест, который обнаруживает еще не выявленную ошибку.

Существуют различные способы тестирования программ.

Тестирование программы как «черного ящика» (стратегия «черного ящика» определяет тестирование с анализом входных данных и результатов работы программы). Критерием исчерпывающего входного тестирования является использование всех возможных наборов входных данных.

Тестирование программы как «белого ящика» заключается в стратегии управления логикой программы, позволяет использовать ее внутреннюю структуру. Критерием выступает исчерпывающее тестирование всех маршрутов и управляющих структур программы.

Разумная и реальная стратегия тестирования - сочетание моделей «черного» и «белого ящиков».

Принципы тестирования:

- описание предполагаемых значений выходных данных или результатов должно быть необходимой частью тестового набора;
- тесты для неправильных и непредусмотренных входных данных следует разрабатывать так же тщательно, как для правильных и предусмотренных;
- необходимо проверять не только делает ли программа то, для чего она предназначена, но и не делает ли она то, что не должна делать;
- нельзя планировать тестирование в предположении, что ошибки не будут обнаружены;
- вероятность наличия необнаруженных ошибок в части программы пропорциональна числу ошибок, уже обнаруженных в этой части;
- тестирование - процесс творческий.

При разработке программ очень полезным бывает метод «ручного тестирования» без ком-

пьютера на основе инспекции и сквозного просмотра (тестирование «всухую»).

Инспекция и сквозной просмотр - это набор процедур и приемов обнаружения ошибок при чтении текста.

Основные типы ошибок, встречающихся при программировании:

- обращения к переменным, значения которым не присвоены или не инициализированы;
- выход индексов за границы массивов;
- несоответствие типов или атрибутов переменных величин;
- явные или неявные проблемы адресации памяти;
- ошибочные передачи управления;
- логические ошибки.

При проектировании процедуры тестирования предусматривают серии тестов, имеющих наивысшую вероятность обнаружения большинства ошибок. Для целей исчерпывающего тестирования создают эквивалентные разбиения входных параметров, причем предусматривают два класса: правильные входные данные и неправильные (ошибочные входные значения). Для каждого класса эквивалентности строят свой тест. Классом эквивалентности тестов можно назвать такое множество тестов, что выполнение алгоритма на одном из них гарантирует аналогичный результат прогона для других.

Особое внимание необходимо уделять тестам на граничных условиях. Граничные условия - это ситуации, возникающие непосредственно на, выше или ниже границ входных и выходных классов эквивалентности (т.е. вблизи границ эквивалентных разбиений). В частности, примерами классов эквивалентных тестов для алгоритма решения квадратного уравнения могут служить следующие классы: множество действительных, отличных от нуля, чисел a, b, c , таких, что $b \cdot b - 4 \cdot a \cdot c < 0$; множество чисел $a = 0$, b и c не равны нулю; $b = 0$, a и c не равны нулю, и т.п.

Сам процесс тестирования может быть пошаговым и/или монолитным. В том и в другом случае используют стратегии нисходящего тестирования, - начиная с верхнего, головного модуля, и затем подключая последовательно другие модули (аппарат заглушек), и восходящего тестирования, начиная с тестирования отдельных модулей.

В процессе отладки программы используют метод грубой силы - использование выводов промежуточных данных по всей программе (трассировка) или использование автоматических средств. Например, в Турбо-Паскале имеется в наличии мощный аппарат автоматической отладки программ (режим DEBUG).

Есть золотое правило программистов - оформляй свои программы в том виде, в каком бы ты хотел видеть программы, написанные другими. К каждому конечному программному продукту необходимо документированное сопровождение в виде помощи (help), файлового текста (readme.txt).

Контрольные вопросы и задания

1. Каковы основные этапы проектирования и разработки программы?
2. Что означает хорошо сформулированная постановка задачи?
3. Назовите методологии проектирования и разработки программ.
4. Как выбрать модель задачи?
5. Что такое тестирование программы?
6. Постройте группу тестов для алгоритма решения системы линейных уравнений.

4.2. ОСНОВНЫЕ ПРИНЦИПЫ РАЗРАБОТКИ И АНАЛИЗА АЛГОРИТМОВ

При построении алгоритма для сложной задачи используют системный подход - использованием декомпозиции (нисходящее проектирование сверху-вниз) и синтеза (программирование снизу-вверх). Как и при разработке структуры любой сложной системы, при формировании алгоритма используют дедуктивный и индуктивный методы. .

При дедуктивном подходе рассматривается частный случай общеизвестных алгоритмических моделей. Здесь при заданных предположениях известный алгоритм приспособляется к условиям решаемой задачи. Например, многие вычислительные задачи линейной алгебры, в частности, нелинейные уравнения, системы алгебраических уравнений и т.п., могут быть решены с использованием известных методов и алгоритмов, для которых существует множество специальных

библиотек подпрограмм, модулей. В настоящее время получили распространение специализированные пакеты, позволяющие решать многие задачи (Mathcad, Eureka, Reduce— Autocad и т.п.).

Индуктивный способ предполагает эвристический системный подход (декомпозиция - анализ - синтез). В этом случае общих и наиболее удачных методов не существует. Возможны некоторые подходы, позволяющие в каждом конкретном случае находить и строить алгоритмы. Методы разработки алгоритмов можно разбить на методы частных целей, подъема, отработки назад, ветвей и границ и т.п.

Одним из системных методов разработки алгоритмов является структурное программирование. Принципы структурной алгоритмизации ранее излагались в гл. 1 (п. 1.8). Повторим их более формально с упором на реализацию в практически программировании.

Структурное программирование основано на использовании блок-схем, формируемых с помощью управляющих структурных элементов. Блок-схема - это ориентированная сеть, у которой могут быть вершины типа изображенных на рис. 3.5.

Выделяют три базовых структурных элемента (управляющие структуры): композицию, альтернативу, итерацию.

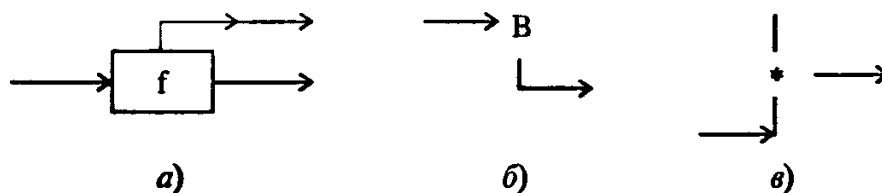


Рис. 3.5. Функциональные (а), предикатные (б) и объединяющие (в) вершины

Композиция - это линейная конструкция алгоритма, составленная из последовательно следующих друг за другом функциональных вершин, рис.3.6.

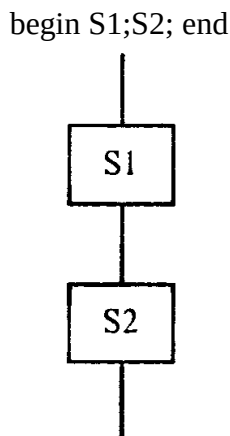


Рис. 3.6. Структура «композиция»

Альтернатива - это конструкция ветвления, имеющая предикатную вершину. Конструкция ветвления в алгоритмах может быть представлена в виде развилки (а), неполной развилки (б) и выбора (в) (рис. 3.7).

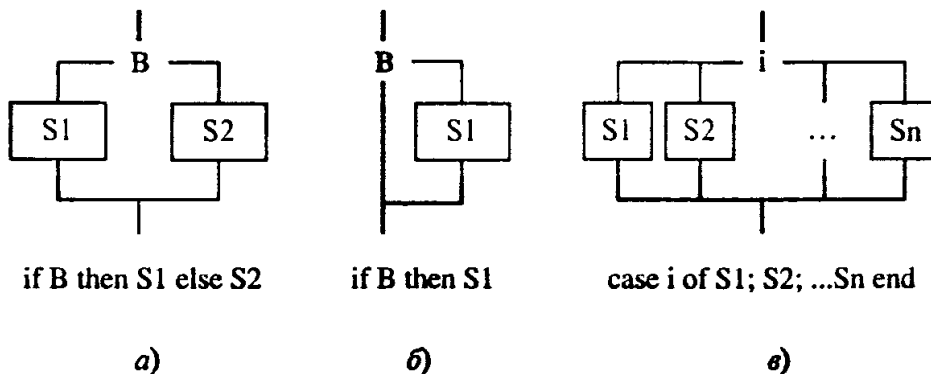


Рис. 3.7. Структура «альтернатива». Здесь В - условие (логическое выражение)

Итерация - это циклическая конструкция алгоритма, которая, вообще говоря, является составной структурой, состоящей из композиции и альтернативы. Итерации могут быть представлены в двух формах: с предусловием (а) и с постусловием (о) (рис.3.8).

Каждая из рассмотренных структур имеет один вход и один выход. Поэтому любая компьютерная программа может быть представлена блок-схемой, сформированной из представленных трех управляющих структур.

Процесс структурного программирования обычно начинается с разработки блок-схемы. Для представления алгоритма в полном и законченном виде, а также

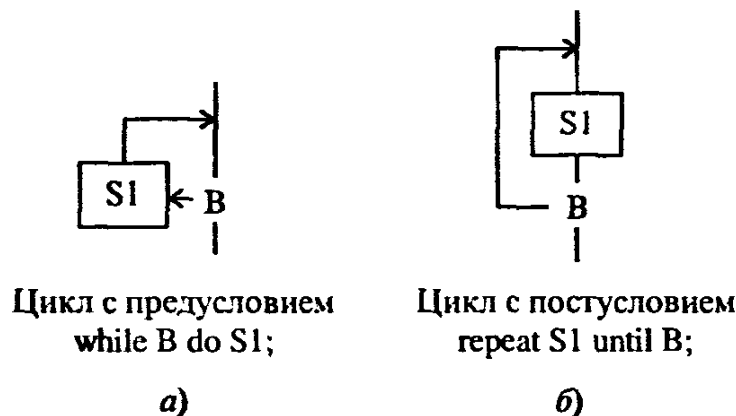


Рис. 3.8. Структура «итерация»

для обозначения связей с окружающей средой добавляют дополнительные структуры ввода-вывода и начала-конца программного блока, модуля, алгоритма:



Заметим, что для начального шага разработки программы чрезвычайно важным и необходимым является определение исходных (ввод) и выходных (вывод) данных задачи. С этого этапа начинается разработка практически любого алгоритма.

Метод разработки программы сверху-вниз предполагает процесс пошагового разбиения алгоритма (блок-схемы) на все более мелкие части до уровня элементарных конструкций, для которых можно составить конкретные команды. Идея структурного программирования сверху-вниз состоит в том, что, если для некоторой функции f существует ее композиция через две другие функции g и h , т.е. $f=h(g(x))$, то проблема разработки алгоритма для f сводится к проблемам разработки алгоритмов для h и g . В структурном программировании сверху-вниз на каждом шаге пытаются текущую функцию выразить как композицию двух (или более) других функций, которые представимы в виде рассмотренных выше управляющих структур.

Для иллюстрации технологии структурного программирования сверху-вниз рассмотрим два примера - сначала простой, затем существенно более сложный.

Пример 1. Технология разработки программы решения квадратного уравнения.

На рис. 3.9 проиллюстрирована пошаговая детализация процесса построения алгоритма. Заметим, что для начального шага разработки программы имеем в качестве исходных данных коэффициенты a, b, c квадратного уравнения $ax^2 + bx + c = 0$, а на выходе - значения двух корней x_1, x_2 .

Пример 2. Рассмотрим более сложный и поучительный пример структурной программирования, известный в литературе как «тур шахматного коня». В задаче необходимо ответить на вопрос, существует ли при заданном положении шахматного коня последовательность его ходов, единожды содержащая все клетки шахматного поля.

Попытка быстро ответить на этот вопрос приводит к перебору всех возможных маршрутов коня. Число вариантов перебора чрезвычайно велико, и поиск нужного маршрута лучше поручить компьютеру.

Одной из эвристических стратегий алгоритма может быть следующая. Начиная с произвольного поля i, j (на рис.3.10 $i = 4, j = 4$), пытаемся пойти на поле $*1$, если невозможно, то на поле

*2; при неудаче - на поле *3 и т.д. по часовой стрелке

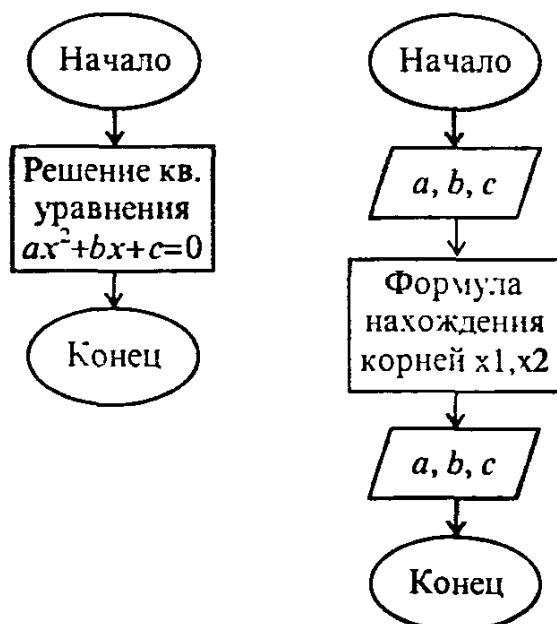


Рис. 3.9. Пошаговая детализация построения алгоритма

(варианты возможных ходов приведены на рисунке справа). Сделав очередной ход на пустую клетку, запишем в нее номер очередного хода и снова осуществляем процедуру поиска нового хода. В случае, когда из очередной клетки невозможно сделать ход, прерываем маршрут и выводим результат в виде таблицы, соответствующей шахматному полю, в которой расставлены ходы коня. Очевидно, что такая стратегия лишь при удаче может дать полный тур коня.

Итак, исходные данные задачи - произвольные начальные координаты коня i, j от 1 до 8. Результат - возможный маршрут коня из заданного поля. Удачным считается маршрут, содержащий все 64 хода, т.е. полный тур коня.

		j							
		1	2	3	4	5	6	7	8
i	1								
	2			*8		*1			
	3		*7				*2		
	4				\$				
	5		*6				*3		
	6			*5		*4			
	7								
	8								

Рис.3.10. Иллюстрация к задаче «тур шахматного коня»

Инициализация доски предполагает задание двумерного массива размером 8×8 с нулевыми элементами. В дальнейшем элемент $a[i,j]$ принимает значения номера очередного хода. Распечатать результат - означает вывести таблицу $a[1..8, 1..8]$. На рис.3.12 показан один из результатов возможного маршрута коня из начального поля $i=1, j=1$.

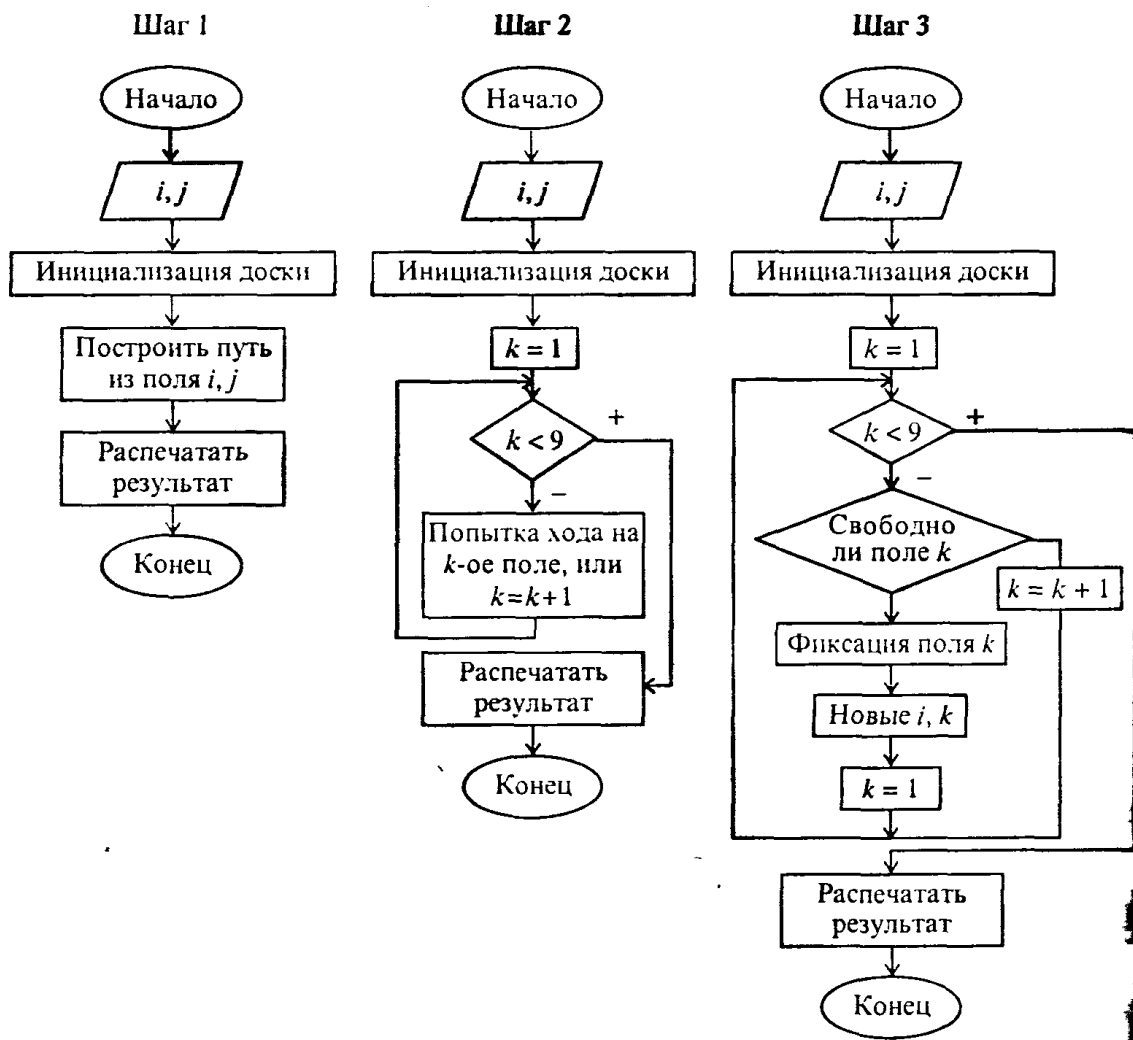


Рис. 3.11. Пошаговая детализация построения алгоритма к примеру 2

	1	2	3	4	5	6	7	8
1	1			34	3	36	19	22
2			2	37	20	23	4	17
3			33		35	18	21	10
4			38		24	11	16	5
5		32			39	26	9	12
6				25		15	6	27
7	31			40	29	8	13	42
8			30		14	41	28	7

Рис. 3.12. Возможный результат маршрута коня из поля (1.1)

Программа 35

```

Program Tur_Konja;
var a: array[1..8,1..8] of integer;
    im, jm :array[1..8] of integer;
    i, j, k, n, inac, jnac: integer;
    inext, jnext: integer;
begin (-----инициализация шахматной доски-----)
  for i:=1 to 8 do for j:=1 to 8 do a[i,j]:=0;
  im[1]:=-2; jm[1]:=1.; im[2]:=-1; jm[2]:=2; im[3]:=1; jm[3]:=2;

```

```

im[4]:=2; jm[4]:=1; im[5]:=2; jm[5]:=-1; im[6]:=1; jm[6]:=-2;
im[7]:=-1; jm[7]:=-2; im[8]:=-2; jm[8]:=-1;
write('введи начальные координаты коня 0<i,j<9: ');
readln(inac,jnac) ;
a[inac,jnac]:=1; i:=inac; j:=jnac; n:=2; k:=1;
while k<=8 do
begin inext:=i+im(k); jnext:=j+jm(k) ;
  if (inext<1) or (inext>8) or (jnext<1) or
    (jnext>8) or (a[inext,jnext]<>0)
  then k:=k+1
  else begin a(inext,jnext):=n; n^n+1; i:=-inext;
    j:=-jnext; k:=1;
  end;
end;
{-----вывод результата прохода-----}
for i:=1 to 8 do
begin writeln; writeln; for j:=1 to 8 do write(a(i,j):2, ' ');
end;
writeln; write('кол-во шагов = ',n-1); readln;
end.

```

Зачастую используют альтернативный процедуре сверху-вниз метод структурного программирования снизу-вверх. По сути мы приходим к конечному результату системным методом. Сначала разбиваем задачу на отдельные блоки (модули) с их связями между собой (декомпозиция), затем, после их разработки, проводим сборку блоков в единую программу (синтез). Принцип снизу-вверх широко распространен среди программистов, которые предпочитают модульный подход, предполагающий максимальное использование стандартных и специализированных библиотек процедур, функций, модулей и объектов.

Задания

1. Используя принцип проектирования сверху-вниз постройте блок-схему и программ для решения системы линейных алгебраических уравнений методом Гаусса.
2. Разработайте алгоритм и программу поиска тура коня по другой стратегии, например, по случайному выбору очередного хода из числа возможных.

4.3. МЕТОДЫ ПОСТРОЕНИЯ АЛГОРИТМОВ, ОРИЕНТИРОВАННЫЕ НА СТРУКТУРЫ ДАННЫХ

Часто на технологию разработки алгоритма влияют структуры данных, используемых в программе. Удачный выбор структур данных позволяет зачастую легко строить эффективные алгоритмы. Методы программирования, в которых такое влияние доминирует, называют методами, ориентированными на структуры данных. Рассмотрим некоторые классы задач, где полезны такие структуры как связные списки, очереди, стеки, деревья.

Сортировка массивов данных, т.е. расположение их элементов в определенном порядке, являясь одной из важнейших прикладных задач при эксплуатации информационных систем, требует больших временных затрат и ресурсов памяти ЭВМ. Легко представить возникающие трудности, когда в массиве данных происходят удаления и внесения новых записей. Обычные подходы заставляют нас осуществлять заново сортировку измененного массива с физическими перестановками записей согласно известным процедурам упорядочивания.

Попробуем проблему решить с помощью линейного связанного списка. Массив преобразуют в двумерный, в котором по второму индексу (целые неотрицательные числа, называемые связями или указателями) располагают номера элементов массива.

Info

Link

1 Петров	3	Линейный связанный список - это конечный набор пар, состоящих из информационной части (Info) и указующей части (Link).
2 Смирнов	4	
3 Алексеев	1	
N Кузнецов	2	

Линейные связанные списки являются эффективной структурой данных для моделирования ситуаций, в которых подвергаются изменениям упорядоченные массивы элементов данных. Особенно важно их использование при процедурах внесения или удаления элементов из середины массива. Когда модификации касаются лишь начала или/и конца, то необходимость в связанных списках отпадает, и становится достаточным использование одномерного исходного массива. Здесь на помощь приходят стеки и очереди.

Пусть, например, задано арифметическое выражение. Требуется определить, правильно ли расставлены в выражении скобки.

Для решения подобных задач используют стековую память (называемую просто «стек»). стек представляет последовательность данных и имеет лишь одну границу для добавления и удаления элементов. В нашем случае в стек помещаются и удаляются скобки.

Первым необходимым условием правильности расстановок скобок является совпадение количества левых и правых скобок. Такой контроль легко осуществить введя счетчик *top*, который при просмотре выражения и обнаружении левой скобки (допустим, что имеем только круглые скобки '(') увеличивается на +1. Если на очередном месте встретилась правая скобка, то значение счетчика уменьшается на 1. Тогда правильность расстановки определяется по итоговому значению *top*.

Программа 36

```

program skobkal; (*проверка скобок по количеству*)
  var top, i, n: integer; slovo: string[100]; skob: string[100];
begin
  write('введи арифметическое выражение: ');
  readln(slovo); n:=length(slovo);
  top:=0; skob:=''; i:=1;
  while (i<=n) do begin
    if slovo[i]='(' then begin top:=top+1; skob:=skob+slovo[i]
end;
    if slovo[i]=')' then begin
      top:=top-1; skob:=skob+slovo[i] end;
    i:=i+1
  end;
  writeln(skob) ;
  if top=0 then write('выражение правильное') i else
write('выражение неправильное');
  readln .
end.

```

Строковая переменная *skob* предназначена "для визуализации всех имеющихся скобок в выражении.

В случае, когда в выражении используются фигурные, квадратные и круглые скобки, задача усложняется тем, что необходим еще контроль соответствия левых и правых скобок. В этой связи удобно использовать стек, в котором помещаются очередные левые скобки. При обнаружении правой скобки из вершины стека извлекается левая скобка, помещенная последней, и проводится их идентификация. Полный текст программы представлен ниже.

Программа 37

```

program skobka2; (*проверка расстановок скобок*) var
  top, i, n: integer;
slovo: string[100];
store: array [1..100] of char; -x: char.; sicob: string[100];
p: boolean;
begin
  write('введи арифметическое выражение: ');
  readln(slovo); n:=length(slovo) ;
  top:=0; p:=true; skob:=''; i:=1;
  while (i<=n)and(p) do
    begin if (slovo[i]='(') or (slovo[i]='[') or (slovo[i]='(')
      then begin top:=top+1; store[top]:=slovo[i];
skob:=skob+slovo[i] end;
    if slovo[i]='}' then begin x:=store[top];
    if x<>'(' then p:=false
      else begin top:=top-1; skob:=skob+slovo[i] end;
    end;
    if slovo[i]=']' then begin x:=store[top] ;
    if x<>'[' then p:=false
      else begin top:=top-1; skob:=skob+slovo[i] end;
    end;
    if slovo[i]=')' then begin x:=store[top] ;
    if x<>'(' then p:=false
      else begin top:=top-1; skob:=skob+slovo[i] end;
    end;
    i:=i+1 end;
    writeln(skob); if top=0 then write('выражение правильное')
      else write('выражение неправильное');
  readln
end.

```

Структура данных «очередь» используется для моделирования систем массового обслуживания: очереди людей в магазинах, транспортных потоков, производственных линий и т.п. Рассмотрим модельную ситуацию с формированием очереди в каком-нибудь учреждении сферы обслуживания, например, в банке.

Пусть задана скорость поступления клиентов в банк и известна скорость обслуживания. Вместо скорости поступления клиентов будем задавать вероятность p их появления в единицу времени. За скорость обслуживания примем число v , соответствующее времени обслуживания одного клиента. Для простоты примем в качестве массива данных о клиентах банка числовой массив со случайными числами из интервала 1..100. Для формирования очереди достаточно ввести две переменные, которые указывают на начало и конец списка данных.

Следующая программа демонстрирует динамику обслуживания очереди.

Программа 38

```

program bank;
uses crt;
type item = integer;
var qq:array[1..100] of item; i, t, v, L, R; integer;
p, x: real; st: string[10];
begin (*начальное состояние очереди*)
  qq[1]:=random(100); qq[2]:=random(100); qq[3]:=random(100);
  L:=1; R:=3; p:=0.6 v:=2; randomize; t:=0;
  repeat
    t:=t+1; x:=random; if x<p then begin R:=R+1;
qq[R]:=random(100);
  end;
end;

```

```

if (t mod v=0) then L:=L+1;
until keypressed or (R>100) ;
(*вывод состояния очереди на момент прерывания*) for
i:=L to R do writeln(qq[i]);
readln;
end.

```

Контрольные задания

1. Постройте программу упорядочивания списка фамилий студентов группы с использованием линейного списка.
2. С помощью стека организуйте алгоритм, который определяет, является ли заданное слово палиндромом («перевертышем»).
3. Придумайте задачу на использование очереди.

4.4. РЕКУРСИВНЫЕ АЛГОРИТМЫ

Изучая в предыдущем разделе язык Паскаль, мы уже использовали понятие рекурсии. Однако, оно столь важно и принципиально, что с ним следует познакомиться детальнее.

Рекурсией называют метод определения или вычисления функции, процедуры или решения задачи посредством той же функции, процедуры и т.д. Рекурсивные алгоритмы широко используют методы частных целей, подъема и отработки назад. На эвристическом уровне рекурсия позволяет эффективно использовать метод проб и ошибок.

Продолжим рассмотрение примера задачи тура шахматного коня из предыдущего раздела. Приведенный там алгоритм строил возможный путь коня по простой стратегии очередного хода на свободное место по принципу часовой стрелки. Однако, он не позволял гарантированно найти полный тур коня. Применим простую эвристическую модель решения задачи - в случае отсутствия возможности очередного хода осуществляется возврат коня на предыдущее поле и возобновление поиска дальнейшего маршрута по другому пути. Подобный процесс называют возвратом (или откатом). Его можно осуществлять по универсальной схеме:

```

procedure RETR;
begin
    инициализация начального хода repeat выбор оче-
редного хода
        if подходит then его запись;
if решение не полное then RETR;
if неудача then стирание хода и возврат на предыдущий until удача or нет хода
end.

```

Подобная рекурсивная процедура и уже известный алгоритм, рассмотренный выше, позволяют построить нужную программу. Ниже представлена программа тура коня для произвольного поля NxN, позволяющая отыскивать полный тур с любого начального положения. Для наглядной иллюстрации процесса поиска в глубину и в ширину с возвратами в программе в комментарные скобки обозначены команды вывода промежуточных результатов.

Программа 39

```

program tur;


---


var i, j, ii, jj, n, nn: integer; q: boolean;
    dx, dy: array[1..8] of integer; h: array[1..8, 1..8] of integer;
    (*рекурсивная процедура - попытка сделать ход*)
procedure try(i, x, y: integer; var q: boolean);
var k, u, v: integer; ql: boolean;
begin
    k:=0; repeat k:=k+1; ql:=false; u:=x+dx[k]; v:=y+dy[k];
        if ( (1<=u) and(u<=n) and (1<=v) and (v<=n) ) and(h[u,v]=0)
            then begin h[u,v]:=i;

```

```

(*для отладки и наблюдения процесса поиска с возвратом*)
for ii:=1 to n do begin for jj:= 1 to n do
    write(h[ii,jj]:5); writeln;
end;
readln;
    if i<nn then begin try(i+1,u,v,ql);
    if not(ql) then h[u,v]:=0
        else ql:=truer;
        end
    else ql:=true
end;
until (ql) or (k=8);
q:=ql
end; (* конец процедуры*)
begin
    dx[1] =2: dx[2]:=1; dx[3]:=-1; dx[4]:=-2; dx[5]:=-2;
    dx[6] =-1: dx[7]:=1; dx[8]:=2; dy[1]:=1; dy[2]:=2;
    dy[3] =2: dy[4]:=1; dy[5]:=-1; dy[6]:=-2;
    dy[7] =-2: dy[8]:=-1;
    write ('введи n: '); readln(n);
    for i =1 to n do for j:=1 to n do h[i,j]:=0;
    write; ('введи i,j : '); readln(i,j); nn:=n*n;
h[i,j]:=1; try(2,i,j,q);
    if q then begin
        for i:=1 to n do begin
            for j:= 1 to n do write(h[i,j]:5);
            writeln;
        end;
    end ' else writeln( 'нет маршрута');
    readln
end.

```

Для $n = 5$ и $n = 6$ алгоритм быстро находит искомые туры коня. Для $n = 8$ время решения может возрасти в несколько десятков раз.

Рассмотрим еще два замечательных рекурсивных алгоритма, позволяющих строить регулярные образы, в конечном счете образующие красивые узоры на экране дисплея. Узор образуется из серии выстраиваемого определенным образом заданного мотива.

Ниже представлена программа, использующая при построении узора кривые Серпинского, рис. 3.13.

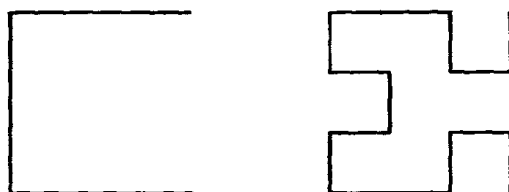


Рис. 3.13. Примеры кривых Серпинского

На рисунке изображены кривые Серпинского S_1 и S_2 первого и второго порядков. Кривую Серпинского S_i можно разбить на 4 части: A_i , B_i , C_i , D_i , которые соединяются четырьмя отрезками. Эти четыре части кривой представляют одну и ту же ломаную, поворачивающуюся каждый раз на 90 градусов. Нетрудно увидеть рекурсивные схемы, по которым ломаные A_i , B_i , C_i , D_i получаются из кривых A_{i-1} , B_{i-1} , C_{i-1} , D_{i-1} , размеры которых при этом сокращаются вдвое:

Ai: A(i-l)	B(i-l) –	D(i-l)	A(i-l)
Bi: B(i-l)	C(i-l)	A(i-l)	B(i-l)
Ci: C(i-l)	D(i-l) –	B(i-l)	C(i-l)
Di: D(i-l)	A(i-l)	C(i-l)	D(i-l)

Векторы, соединяющие отдельные элементы кривых, образуют с осью абсцисс углы, кратные $\pi/4$; средние векторы во всех схемах имеют длину, в два раза большую, чем крайние. Для построения вектора длины U под углом $T*\pi/4$ к оси абсцисс в программе описана процедура `linep(T,U:integer)`:

Программа 40

```

program serpinsk;
uses crt, graph;
const sq=512;
var i, x0, y0, x, y, t, u, gd, gm: integer; ch: chart;

procedure linep(t,u:integer);

    var xl, yl:integer;
begin x:=x+round(u*cos(t*pi/4)) ;
      y:=y-round(u*sin(t*pi/4)); lineto(x,y) ;
end;

procedure b(i:integer); forward;
procedure c(i:integer); forward;
procedure d(i:integer); forward;

procedure a(i:integer);
begin
    if i>0 then begin a(i-1); linep(7,u); b(i-1); linep(0,2*u);
                   d(i-1); linep(1,u); a(i-1)
    end end; •

procedure b;
begin
    if i>0 then begin b(i-1); linep(5,u); c(i-1); linep(6,2*u);
                   a(i-1); linep(7,u); b(i-1)
    end end;

procedure c;
begin
    if i>0 then begin c(i-1); linep(3,u); d(i-1); linep(4,2*u) ;
                   b(i-1); linep(5,u); c(i-1)
    end end;

procedure d;
begin
    if i>0 then begin d(i-1); linep(1,u); a(i-1); linep(2,2*u);
                   c(i-1); linep(3,u); d(i-1) end. end;

begin gd:=0; initgraph(gd, gm, ' ');
      u:=sq div 4; x0:=320; y0:=128; i:=0;
      repeat
          i:=i+1; x0:=x0-u; u:=u div 2; y0:=y0-u;
          x:=x0; y:=y0; setcolor(2*i);
          moveto(x,y); a(i); linep(7,u); b(i);

```

```

        linep(5,u); c(i); linep(3,u);
        d(i); linep(1,u); delay(2000) until
        i=5;
        settextstyle(0,0,1) ;
        outtextxy(200, 470, 'КРИВЫЕ СЕРПИНСКОГО S1 - S5');
        readln; closegraph
end.

```

В 1891 г. Д. Гильберт открыл серию рекурсивных кривых, которые получили название кривых Гильберта. Кривая Гильберта H_i , подобно кривым Серпинского, может быть получена из четырех экземпляров кривой H_{i-1} вдвое меньшего размера, повернутых должным образом и соединенных отрезками. Ниже приводится программа, рисующая узор из шести кривых Гильберта.

Программа 41

```

program hilbert;
  uses crt,graph;
const sq=448;
  var i,x0,y0,x,y,t,u,gd,gm:integer; ch:char;
  procedure linep(t,u:integer);
    var xl,yl:integer;
  begin x:=x+round(u*cos(t*pi/4)); y:=y-round(u*sin(t*pi/4)) ;
    lineto (x,y) ;
  end;
  procedure b(i:integer); forward;
  procedure c(i:integer); forward;
  procedure d(i:integer); forward;
  procedure a(i:integer);
  begin
    if i>0 then begin d(i-1); linep(4,u); a(i-1); linep(6,u);
      a(i-1); linep(0,u); b(i-1)
    end end;

  procedure b;
  begin
    if i>0 then begin c(i-1); linep(2,u); b(i-1); linep(0,u);
      b(i-1); linep(6,u); a(i-1)
    end end;

  procedure c;
  begin
    if i>0 then begin b(i-1); linep(0,u); c(i-1); linep(2,u);
      c(i-1); linep(4,u); d(i-1)
    end end;

  procedure d;
  begin
    if i>0 then begin a(i-1); linep(6,u); d(i-1); linep(4,u);
      d(i-1); linep(2,u); c(i-1)
    end end;

  begin gd:=0; initgraph(gd, gm, ' '); x0:=320; y0:=240; u:=sq;
i:=0;
  repeat
    i:=i+1; u:=u div 2;
    x0:=x0+(u div 2); y0:=y0-(u div 2) ;
    x:=x0; y:=y0; setcolor(2*i);
  moveto(x,y); a(i); delay(2000) until i=6;
  settextstyle(0,0,1) ;

```



```

outtextxy(220,470, 'КРИВЫЕ ГИЛЬБЕРТА H1 - H6');
readin; closegraph
end.

```

Контрольные задания

1. Разработайте алгоритм и программу расстановки ферзей на шахматном поле таким образом, чтобы ни один из них не бил другого,
2. Разработайте программу игры «Ханойские башни».
3. Предложите другие модификации алгоритма полного тура коня.

4.5. ВАЖНЕЙШИЕ НЕВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ (ПОИСК И СОРТИРОВКА)

Одними из важнейших процедур обработки структурированной информации являются **сортировка и поиск**. Сортировкой называют процесс перегруппировки заданной последовательности (кортежа) объектов в некотором определенном порядке. Определенный порядок (например, упорядочение в алфавитном порядке, по возрастанию или убыванию количественных характеристик, по классам, типам и т.п.) в последовательности объектов необходим для удобства работы с этими объектами. В частности, одной из целей сортировки является облегчение последующего поиска элементов в отсортированном множестве. Под поиском подразумевается процесс нахождения в заданном множестве объекта, обладающего свойствами или качествами задаваемого априори эталона (или шаблона).

Очевидно, что с отсортированными (упорядоченными) данными работать намного легче, чем с произвольно расположенными. Упорядоченные данные позволяют эффективно их обновлять, исключать, искать нужный элемент и т.п. Достаточно представить, например, словари, справочники, списки кадров в неотсортированном виде и сразу становится ясным, что поиск нужной информации является труднейшим делом, если не невозможным.

Существуют различные алгоритмы сортировки данных. И понятно, что не существует универсального, наилучшего во всех отношениях алгоритма сортировки. Эффективность алгоритма зависит от множества факторов, среди которых можно выделить основные:

- числа сортируемых элементов;
- степени начальной отсортированности (диапазона и распределения значений сортируемых элементов);
- необходимости исключения или добавления элементов;
- доступа к сортируемым элементам (прямого или последовательного).

Принципиальным для выбора метода сортировки является последний фактор. Если данные **МОГУТ** быть расположены в оперативной памяти, то к любому элементу возможен прямой доступ. Удобной структурой данных в этом случае выступает *массив* сортируемых элементов. Если данные размещены на внешнем носителе, то к ним можно обращаться лишь последовательно. В качестве структуры подобных данных можно взять *файловый тип*.

В этой связи выделяют сортировку двух классов объектов: массивов (внутренняя сортировка) и файлов (внешняя сортировка).

Процедура сортировки предполагает, что при наличии некоторой упорядочивающей функции F расположение элементов исходного множества меняется таким образом, что

$$a_1, a_2 \dots a_n \rightarrow a_{k1}, a_{k2} \dots a_{kn} \\ F(a_{k1}) < F(a_{k2}) < F(a_{kn})$$

где знак неравенства понимается в смысле того порядка, который установлен в сортируемом множестве.

Поиск и сортировка являются классическими задачами теории обработки данных, решают эти задачи с помощью множества различных алгоритмов. Рассмотрим наиболее популярные из них.

Поиск. Для определенности примем, что множество, в котором осуществляется поиск, задано как массив

var a:array[0..N] of item;

где item - заданный структурированный тип данных обладающий хотя бы одним полем (ключом), по которому необходимо проводить поиск.

Результатом поиска, как правило, служит элемент массива, равный эталону, или отсутствие такового.

Линейный поиск. Процедура заключается в простом последовательном просмотре всех элементов массива и сравнении их с эталоном X.

i:=0;

while (i<=N)and(a[i]<>X) do i:=i+1 end.

Часто бывает целесообразнее осуществлять поиск с барьером, вводя дополнительно граничный элемент массива a[N+1]:

a[N+1]:=X;i:=0;

while a[i]<>X do i:=i+1 end.

Равенство i = N + 1 означает, что совпадений не было, т.е. что эталонный элемент отсутствует.

Попытайтесь разобраться в чем различие представленных конструкций. Приведем пример программы поиска эталона x в массиве a[0..n].

Программа 42

```
program poisk1; (*линейный поиск*) const
N=8;
type item= integer;
var a : array[0..n] of item; i :integer; x : item;
begin
  (*задание искомого массива*) for i:=0
to N do
  begin writet'Введи элемент a[ ',i, ']= '); readln(a[i]);
end;
writeln; write('введи эталон x= '); readln(x);
(* линейный поиск*)
  i:=0; while (i<=N)and(a[i]<>X) do begin i:=i+1
end;
(*вывод результата*)
  if i<=N then write( 'найден элемент на ',i, ' месте ')
  else write( 'такого элемента в массиве нет ') ;
  readln
end.
```

Поиск делением пополам. В большинстве случаев процедура поиска применяется к упорядоченным данным (телефонный справочник, библиотечные каталоги и пр.). В подобных ситуациях эффективным алгоритмом является поиск делением пополам. В этом методе сравнение эталона X осуществляется с элементом, расположенным в середине массива и в зависимости от результата сравнения (больше или меньше) дальнейший поиск проводится в левой или в правой половине массива.

L:=0; R:=N; while L<R do

begin

m:=(L+R) div 2;

if a[m]<X then L:=m+1 else R:=m;

end;

Например, пусть эталонный ключ x=13, а в массиве имеются следующие элементы:

a[0]=1; a[1]=3; a[2]=4; a[3]=7; a[4]=8; a[5]=9; a[6]=13; a[7]=20; a[8]=23.

Бинарный процесс поиска показан ниже:

1	3	4	7	8	9	13	20	23 - элементы массива
0	1	2	3	4	5	6	7	8- порядковые номера элементов
L				m				R
L				m				R
				a[m]=x				=>поиск закончен и m = 6

Программа поиска представлена ниже.

Программа 43

```

program poisk2; (*поиск делением пополам*)

    const N=8;
    type item= integer;
    var a: array[0..n] of item; i, L, R, m:integer; x: item; f:
boolean;
begin
    (*задание искомого массива*)
    for i:=0 to N do
    begin write( 'введи элемент a[' ,i, ']= '); readln(a[i])
        end;
    writeln; write( 'введи эталон x= '); readln(x);
    (*бинарный поиск*)
    L:=0; R:=N; f:=false;
    repeat m:=(L+R) div 2; if a[m]=X then f:=true;
    if a[m]<X then L:=m+1
        else R:=m;
        writeln(m,L,R);
    until (L>=R) or (f);
    (*вывод результата*)
    if f then write('найден элемент на ',m, ' месте')
        else write('такого элемента в массиве нет ');
    readln
end.

```

Сортировка массивов. Как и в случае поиска определим массив данных:

```
var a: array [0.. N] of item
```

Важным условием сортировки массива большого объема является экономное использование доступной памяти. В прямых методах сортировки осуществляется принцип перестановки элементов «на том же месте». Ниже рассмотрим три группы сортировок: с помощью включения, выбора и обмена.

Сортировка с помощью включения Кто играл в карты, процедуру сортировки включениями осуществлял многократно. Как правило, после раздачи карт игрок, держа карты веером в руке, переставляет карты с места на место стремясь их расположить по мастям и рангам, например, сначала все тузы, затем короли, дамы и т.д. Элементы (карты) мысленно делятся на уже «готовую последовательность» и неправильно расположенную последовательность. Теперь на каждом шаге, начиная с $i = 2$, из неправильно расположенной последовательности извлекается очередной элемент и перекладывается в готовую последовательность на нужное место.

```

for i:=2 to N do begin
    x:=a[i];
    <включение x на соответствующее место готовой последо-
    вательности a[1],...,a[i]>
end

```

Поиск подходящего места можно осуществить одним из методов поиска в массиве, описанным выше. Затем x либо вставляется на свободное место, либо сдвигает вправо на один индекс всю левую сторону. Схематично представим алгоритм для конкретного примера:

Исходные элементы	23	34	12	13	9

i=2	23	34	12	13	9
i=3	12	23	34	13	9
i=4	12	13	23	34	9
i=5	9	12	13	23	34

В алгоритме поиск подходящего места осуществляется как бы просеиванием x при движении по последовательности и сравнении с очередным $a[j]$. Затем x либо вставляется на свободное место, либо $a[j]$ сдвигается вправо и процесс как бы «уходит» влево.

Программа 44

```

program sortirov)ca_1;
(*сортировка включением по линейному поиску*) const N=5;
type item= integer;
var a: array[1..n] of item; i, j: integer; x: item;
begin (*задание искомого массива*)
    for i:=1 to N do begin write('введи элемент a[' , i, ']=');
readln(a[i]) end;
    for i:=1 to N do begin write(a[i], ' ');
end;
writeln;
(*алгоритм сортировки включением*) .for i:=2 to n do begin
    x:=a[i]; j:=i; a[0]:=x; (*барьер*)
    while x<a[j-1] do
        begin
            a[j]:=a[j-1]; j:=j-1;
end;
    a[j]:=x; .
    (for k:=1 to n do write(a[k,1], ' ') end; writeln;) end;
(*вывод отсортированного массива*) for
i:=1 to N do begin .
    write(a[i], ' ');
end;
readln;
end.

```

В рассмотренном примере программы для анализа процедуры пошаговой сортировки можно рекомендовать использовать трассировку каждого прохода по массиву с целью визуализации его текущего состояния. В тексте программы в блоке непосредственного алгоритма сортировки в фигурных скобках находится строка, которая при удалении скобок выполнит требуемое (параметр k необходимо описать в разделе переменных - var k:integer). Во всех последующих программах сортировки легко осуществить подобную процедуру.

Вернемся к анализу метода прямого включения. Поскольку готовая последовательность уже упорядочена, то алгоритм улучшается при использовании алгоритма поиска делением пополам. Такой способ сортировки называют **методом двоичного включения**.

Программа 45

```

program sortirovka_2;
(*сортировка двоичным включением*) const N=5;
type item= integer;
var a: array(1..n] of item; i, j, m, L, R: integer; x: item;
begin
    (*задание элементов массива*) for i:=1 to
N do
        begin write('Введи элемент a[' , i, ']= '); readln(a[i]) ;
end;
    for i:=1 to N do

```

```

begin write (a[i], ' ');
end;
writeln;
(*алгоритм сортировки двоичным включением*)
for i:=2 to n do begin
  x:=a[i]; L:=1; R:=i;
while L<R do begin
  m:=(L+R) div 2; if a[m]<=x then L:=m+1 else R:=m;
end;
  for j:=i downto R+1 do a[j]:=a[j-1];
  a[R]:=x;
end;
(* вывод отсортированного массива*)
for i:=1 to N do
  begin write(a[i], ' ');
end; , readln;
end.

```

Один из вариантов *улучшенной сортировки включением* был предложен Д.Шеллом. Его метод предполагает сначала отдельную группировку и сортировку элементов, отстоящих друг от друга на некотором расстоянии, например 4 (четвертная сортировка), после первого прохода перегруппировку элементов таким образом, чтобы каждый элемент группы отстоял от другого на 2 номера, после двойной сортировки на третьем проходе одинарную (обычную) сортировку.

Исходные элементы	44	5	1	4	9	1	6
		5	2	2	4	8	7
Четвертная сортировка	44	1	6	4	9	5	1
		8		2	4	5	2
Двойная сортировка	6	1	1	4	4	5	9
		8	2	2	4	5	4
Одинарная сортировка	6	1	1	4	4	5	6
		2	8	2	4	5	7
							4

Каждая из сортировок основывается на алгоритме прямого включения и, соответственно, должна программироваться аналогично. Если для условия окончания поиска использовать барьер, а их необходимо ставить для каждой из сортировок, то необходимо расширить границы массива на несколько компонентов (барьеров) влево, т.е. использовать массив $a[-r..n]$, где r - количество сортировок.

Сортировка с помощью прямого выбора. Алгоритм прямого выбора является одним из распространенных в силу своей простоты. Сначала определяют минимальный элемент среди всех элементов массива, затем его меняют местами с первым. Далее процесс повторяется с той лишь разницей, что минимальный ищется со второго и меняется со вторым и т.д.

1	2	3	4	5	
12	15	17	11	13	i=2, min= 11
11	15	17	12	13	i=3.min=12
11	12	17	15	13	i=4, min=13
11	12	13	15	17	i=5,min=15.

Программа 46

```

program sortirovka_3;
(*улучшенная сортировка включением - сортировка Шелла*)
const N=8; t=4;
type item= integer;
var a: array[-9..n] of item; i, j, k, s :integer; x: item;
m: 1..t; h :array [1..t] of integer;

```

```

begin
  (*задание искомого массива*)
  for i:=1 to N do
    begin write('введи элемент a[' , i , ']=') readln(a[i])
    end;
  for i:=1 to N do begin write(a[i], ' ');
    end;
  writeln;
  (*алгоритм Шелла*)
  h[1]:=9; h[2]:=5; h[3]:=3; h[4]:=1;
  for m:=1 to t do
    begin k:=h[m]; s:=-k; (*барьеры для каждого шага*)
      for i:=k+1 to n do
        begin x:=a[i], j:=i-k; if s=0 then s:=-k;- s:=s+1;
          a[s]:=x; while x<a[j] do begin a[j+k]:=a[j]; j:=j-k;
            end;
          a[j+k]:=x
        end;
      end;
    end;
  (*вывод отсортированного массива*)
  for i:=1 to N do begin write(a[i], ' ');
    end;
  readln;
end.

```

Программа 47

```

program sortirovka 4;
  (*сортировка прямым выбором*)
  const N=5;
  type item= integer;
  var a: array[1..n] of item; i, j, k: integer; x: item;
begin
  (*задание искомого массива*)
  for i:=1 to N do
    begin write('введи элемент a[' , i , ']='); readln(a[i]);
    end;
  for i:=1 to N do begin write(a[i], ' ');
    end;
  writeln;
  (*алгоритм прямого выбора*)
  for i:=1 to n-1 do
    begin k:=i; x:=a[i]; (*поиск наименьшего элемента*)
      for j:=i+1 to n do (*и его индекса из a[i]...a[n]*)
        if a[j]<x then begin k:=j; x:=a[k]
          end;
        a[k]:=a[i]; a[i]:=x;
      end;
    end;
  (*вывод отсортированного массива*)
  for i:=1 to N do begin write(a[i], ' ');
    end;
  readln;
end.

```

Улучшенный метод сортировки выбором с помощью дерева. Метод сортировки прямым выбором основан на поисках наименьшего элемента среди неготовой последовательности. Уси-

лить метод можно запоминанием информации при сравнении пар элементов. Этого добиваются определением в каждой паре меньшего элемента за $n/2$ сравнений. Далее $n/4$ сравнений позволит выбрать меньший из пары уже выбранных меньших и т.д. Получается двоичное дерево сравнений после $n-1$ сравнений у которого в корневой вершине находится наименьший элемент, а любая вершина содержит меньший элемент из двух приходящих к ней вершин. Одним из алгоритмов, использующих структуру дерева, является сортировка с помощью пирамиды (Дж.Вилльямс). Пирамида определяется как последовательность ключей $h_L \dots h_R$, такая, что *

$$h_i \leq h_{2i} \text{ и } h_i \leq h_{2i+1}, \text{ для } i=L, \dots, R/2.$$

Другими словами пирамиду можно определить как двоичное дерево заданной высоты h , обладающее тремя свойствами:

- каждая конечная вершина имеет высоту h или $h-1$;
- каждая конечная вершина высоты h находится слева от любой конечной вершины высоты $h-1$;
- значение любой вершины больше значения любой следующей за ней вершины. Рассмотрим пример пирамиды, составленной по массиву

27 9 14 8 5 11 7 2 3.

У пирамиды p вершин, их значения можно разместить в массиве a , но таким образом, что следующие за вершиной из $a[i]$ помещаются в $a[2i]$ и $a[2i+1]$. Заметим, что $a[6]=11, a[7]=7$, а они следуют за элементом $a[3]=14$ (рис.3.14).

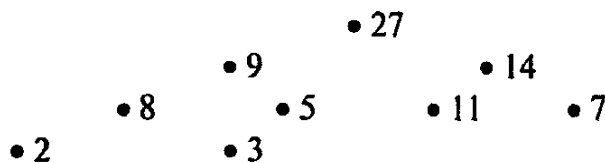


Рис. 3.14. Пирамида

Очевидно, что если $2i > n$, тогда за вершиной $a[i]$ не следуют другие вершины, и она является конечной вершиной пирамиды.

Процесс построения пирамиды для заданного массива можно разбить на четыре этапа:

- 1) меняя местами $a[1]$ и $a[p]$, получаем 3 9 14 8 5 11 7 2 27;
- 2) уменьшаем n на 1, т. е. $p=n-1$, что эквивалентно удалению вершины 27 из дерева;
- 3) преобразуем дерево в другую пирамиду перестановкой нового корня с большей из двух новых, непосредственно следующих за ним вершин, до тех пор, пока он не станет больше, чем обе вершины, непосредственно за ним следующие;
- 4) повторяем шаги 1, 2, 3 до тех пор, пока не получим $p=1$.

Для алгоритма сортировки нужна процедура преобразования произвольного массива в пирамиду (шаг 3). В ней необходимо предусмотреть последовательный просмотр массива справа налево с проверкой одновременно двух условий: больше ли $a[i]$, чем $a[2i]$ и $a[2i+1]$.

Полный текст программы приведен ниже.

Программа 48

```

program sortirovka_5;
  (*улучшенная сортировка выбором - сортировка с помощью дерева*) const
    N=8;

  type item= integer;
  var a : array(1..n) of item; k, L, R: integer; x: item;
  procedure sift(L,R:integer);
  var i, j: integer; x,y: item;
  begin i:=L; j:=2*L; x:=a[L]; if (j<R) and (a[j]<a[j+1]) then j:=j+1;
  while (j<=R)and(x<a[j]) do begin y:=a[i]; a[i]:=a[j];
  a[j]:=y a[i]:=a[j]; i:=j; j:=2*j;
  if (j<R)and(a[j]<a[j+1]) then j:=j+1;
  end;
  end;
end;
```

```

begin
  (*задание искомого массива*) for k:=1 to N do begin
    write('введи элемент a[' ,k, ']=');
    readln(a[k]) ;
  end;
  for k:=1 to N do begin write(a[k], ' ');
  end;
  writeln;
  (*алгоритм сортировки с помощью дерева*) (*построение пирамиды*)
  L:=(n div 2) +1; R:=n; while L>1 do begin L:=L-1; SIFT(L,R);
  end;
  (*сортировка*) while R>1 do begin x:=a[1]; a[1]:=a[R];
a[R]:=x;
  R:=R-1; SIET(1,R);
  end;
  (*вывод отсортированного массива*) for k:=1 to N do
begin write(a[k], ' ');
  end;
  readln;
end.

```

Сортировка с помощью обменов. Характерной чертой алгоритмов сортировки с помощью обмена является обмен местами двух элементов массива после их сравнения друг с другом. В так называемой «пузырьковой сортировке» проводят несколько проходов по массиву, в каждом из которых повторяется одна и та же процедура: сравнение двух последовательно стоящих элементов и их обмен местами в порядке меньшинства (старшинства) Подобная процедура сдвигает наименьшие элементы к левому концу массива. Название этого алгоритма связано с интерпретацией элементов как пузырей в сосуде с водой, обладающих весом соответствующего элемента (при этом массив надо представлять в вертикальном положении). При каждом проходе пузырьки всплывают до своего уровня.

Программа 49

```

program Sortirovka_6;
(*сортировка прямым обменом - пузырьковая сортировка*)
const N=5;
  type item= integer; var a: array(1..n) of item; i, j: integer;
x: item;
begin (*задание искомого массива*)
  for i:=1 to N do begin write('введи элемент a[' ,i, ']= ');
  readln(a[i]);
  end;
  for i:=1 to N do begin write(a[i], ' '); „
  end;
  writeln;
  (*алгоритм пузырьковой сортировки*) for i:=2 to n do for j:=n downto i do
begin
  if a[j-1]>a[j] then begin x:=a [j-1] ;a [j-1] :=a[j]; a[j]:=x;
  1 end;
  end;
  (*вывод отсортированного массива*) for i:=1 to N do
begin write(a[i], ' ');
  end;
  readln;
end.

```

Представленную программу можно легко улучшить, если учесть, что если после очередного прохода перестановок не было, то последовательность элементов уже упорядочена, т.е. продолжать проходы не имеет смысла. Читатель без труда сможет внести коррективы в программу, используя логическую переменную, которая контролировала бы факт обмена.

Если чередовать направление последовательных просмотров, алгоритм улучшается. Такой алгоритм называют «шейкерной» сортировкой.

Программа 50

```

program sortirovka_7;
(*сортировка прямым обменом - шейкерная сортировка*) const N=5;
type item= integer;
var a: array[1..n] of item; i, j, k, L, R: integer; x: item;
begin (*задание искомого массива*)
    for i:=1 to N do begin write('введи элемент a(',i,']=');
        readln(a[i]);
    end;
    for i:=1 to N do begin write(a[i], ' end;
        writeln;
    end;
    (*алгоритм шейкерной сортировки*) L:=2; R:=n; k:=n;
repeat
    for j:=R downto L do begin
        if a[j-1]>a[j] then begin x:=a[j-1];a[j-1]:=a[j];
            a[j]:=x; k:=j
        end;
    end;
    L:=k+1;
    for j:=L to R do begin
        if a[j-1]>a[j] then begin x:=a[j-1];
            a[j-1]:=a[j]; a[j]:=x; k:=j end;
    end;
    R:=k-1;
until L>R;
    (*вывод отсортированного массива*)
    for i:=1 to N do
        begin write(a[i], ' ');
            end; readln;
end.

```

Пузырьковая сортировка является не самой эффективной, особенно для последовательностей, у которых «всплывающие» элементы находятся в крайней правой стороне. В улучшенной (быстрой) пузырьковой сортировке предлагается производить перестановки на большие расстояния, причем двигаться с двух сторон. Идея алгоритма заключается в сравнении элементов, из которых один берется слева ($i = 1$), другой -справа ($j = n$). Если $a[i] \leq a[j]$, то устанавливают $j = j - 1$ и проводят следующее сравнение. Далее уменьшают j до тех пор, пока $a[i] > a[j]$. В противном случае меняем их местами и устанавливаем $i = i + 1$. Увеличение i продолжаем до тех пор, пока не получим $a[i] > a[j]$. После следующего обмена опять уменьшаем j . Чередую уменьшение j и увеличение i , продолжаем этот процесс с обоих концов до тех пор, пока не станет $i = j$. После этого этапа возникает ситуация, когда первый элемент занимает ему предназначенное место, слева от него младшие элементы, а справа - старшие.

Далее подобную процедуру можно применить к левой и правой частям массива и т.д. Очевидно, что характер алгоритма рекурсивный. Для запоминания ведущих левого и правого элементов в программе необходимо использовать стек.

Программа 51

```

program sortirovka_8;
(*улучшенная сортировка разделением - быстрая сортировка с рекур-
сией*) const N=8;
type item= integer;
var a: array[1..n] of item; i: integer;
procedure sort(L,R: integer);
var i, j :• integer; x, y: item;
begin
    i:=L; j:=R; x:=a[(L+R) div 2];

```

```

repeat
    while a[i]<x do i:=i+1; while x<a[j] do j:=j-1;
if i<=j then begin y:=a[i]; a[i]:=a[j];
a[j]:=y; i:=i+1; j:=j-1;
end;
until i>j ;
    if L<j then SORT(L,j); if i<R then SORT(i,R); ' end;
begin , . (*задание искомого массива*) for i:=1 to N do begin
    write("Введи элемент a['i, ']='");
    readln(a[i]);
end;
for i:=1 to N do begin write(a[i], ' ');
    end;
writeln;
(*алгоритм быстрой сортировки*) SORT(l,n);
(*рекурсивная процедура*) (*вывод отсортированного массива*)
for i:=1 to N do begin write(a[i], ' ');
end;
readln;
end.

```

Сортировка файлов. Главная особенность методов сортировки последовательных файлов в том, что при их обработке в каждый момент непосредственно доступна одна компонента (на которую оказывает указатель). Чаще процесс сортировки протекает не в оперативной памяти, как в случае с массивами, а с элементами на внешних носителях («винчестере», диске и т.п.).

Понять особенности сортировки последовательных файлов на внешних носителях позволит следующий пример.

Предположим, что нам необходимо упорядочить содержимое файла с последовательным доступом по какому-либо ключу. Для простоты изучения и анализа сортировки условимся, что файл формируем мы сами, используя как и в предыдущем разделе некоторый массив данных. Его же будем использовать и для просмотра содержимого файла после сортировки. В предлагаемом ниже алгоритме необходимо сформировать вспомогательный файл, который позволит осуществить следующую процедуру сортировки. Сначала выбираем из исходного файла первый элемент в качестве ведущего, затем извлекаем второй и сравниваем с ведущим. Если он оказался меньше, чем ведущий, то помещаем его во вспомогательный файл, в противном случае во вспомогательный файл помещается ведущий элемент, а его замещает второй элемент исходного файла. Первый проход заканчивается, когда аналогичная процедура коснется всех последовательных элементов исходного файла. Ведущий элемент заносится во вспомогательный файл последним. Теперь необходимо поменять местами исходный и вспомогательный файлы. После *nil* проходов в исходном файле данные будут размещены в упорядоченном виде.

Программа 52

```

program sortirovka_faila_1;
    (сортировка последовательного файла) const N=8;
    type item= integer;
    var a: array[1..n] of item; i,k: integer; x,y: item;
    f1,f2: text; (file of item);
begin
    (задание искомого массива) for i:=1 to N do begin write('введи
элемент a[ 'i, ']='');
        readln(a[i]);
    end;
    writein; assign(f1, 'dat1.dat'); rewrite(f1);
    assign(f2, 'dat2.dat'); rewrite(f2);
    (формирование последовательного файла) for i:=1 to N do begin
        writein(f1,a[i]);
    end;
    (алгоритм сортировки с использованием вспомогательного файла) for
        k:=1 to (n div 2) do

```

```

begin (извлечение из исходного файла и запись во вспомогательный)
reset(f1); readin(f1,x);

for i:=2 to n do begin readln(f1,y);
if x>y then writein(f2,y) else begin writein(f2,x); x:=y;
end;
end;
writein(f2,x) ;
(извлечение из вспомогательного файла и запись в исходный) rewrite(f1);
reset(f2); readin(f2,x);

for i:=2 to n do begin readin(f2,y);
if x>y then writein(f1,y) else begin writein(f1,x); x:=y;
end;
end;
writeln(f1,x); rewrite(f2);
end;
(вывод результата) reset(f1);
for i:=1 to N do readin(f1,a[i]);
for i:=1 to N do begin write(a[i], ' ');
end;
close(f1); close(f2); readin;
end.

```

По сути можно в программе обойтись без массива $a[1..n]$. В качестве упражнения попробуйте создать программу, в которой не используются массивы.

Многие методы сортировки последовательных файлов основаны на процедуре *слияния*, означающей объединение двух (или более) последовательностей в одну, упорядоченную с помощью повторяющегося выбора элементов (доступных в данный момент). В дальнейшем (чтобы не осуществлять многократного обращения к внешней памяти), будем рассматривать вместо файла массив данных, обращение к которому можно осуществлять строго последовательно. В этом смысле массив представляется как последовательность элементов, имеющая два конца, с которых можно считывать данные. При слиянии можно брать элементы с двух концов массива, что эквивалентно считыванию элементов из двух входных файлов.

Идея слияния заключается в том, что исходная последовательность разбивается на две половины, которые сливаются вновь в одну упорядоченными парами, образованными двумя элементами последовательно извлекаемых из этих двух подпоследовательностей. Вновь повторяем деление и слияние, но упорядочивая пары, затем четверки и т.д. Для реализации подобного алгоритма необходимы два массива, которые поочередно (как и в предыдущем примере) меняются ролями в качестве исходного и вспомогательного.

Если объединить эти два массива в один, разумеется двойного размера, то программа упрощается. Пусть индексы i и j фиксируют два входных элемента с концов исходного массива, k и L - два выходных, соответствующих концам вспомогательного массива. Направлением пересылки (сменой ролей массивов) удобно управлять с помощью булевской переменной, которая меняет свое значение после каждого прохода, когда элементы $a_l, \dots, a_r$ движутся на место $Op+^h \dots, a^h$ и наоборот. Необходимо еще учесть изменяющийся на каждом проходе размер объединяемых упорядоченных групп элементов. Перед каждым последующим проходом размер удваивается. Если считать, что количество элементов в исходной последовательности не является степенью двойки (для процедуры разделения это существенно), то необходимо придумать стратегию разбиения на группы, размеры которых q и $г$ могут не совпадать с ведущим размером очередного прохода. В окончательном виде алгоритм сортировки слиянием представлен ниже.

Программа 53

```

program sortirovka_faila_2;
(сортировка последовательного файла слиянием) const N=8;
type item= integer; var a: arrayd. ,2*n] of item;
i, j, k, L, t, h, m, p, q, ^r: integer; f: boolean;
begin
(задание искомого массива)
for i:=1 to N do begin write( 'введи элемент a[ ',i, ']=');
readln(a[i]) ;

```

```

end;
writein;
(сортировка слиянием) f:=true; p:=1;
repeat
    h:=1; t^n; if f then begin
        i:=1; j:-n;k:=n+1; L:=2*n end else begin k:=1;
        L:=n;i:=n+1; j:-2*n
    end; . repeat
        if m>=p then q:=p else q:=m; m:=m-q;
if m>=p then r:=p else r:=m; m:=m-r;
while (q<>0) and (r<>0) do begin
    if a[i]<a[j] then begin a[k]:=a[i]; k:=k+h;
    i:=i+1;q:=q-1
    end else
    begin a[k]:=a[j]; k:=k+h; j:=j-1;r:=r-1 end;
end;
while r>0 do begin a[k]:=a[j]; k:=k+h; j:=j-1; r:=r-1;
end;
while q>0 do begin
    a[k]:=a[i]; k:=k+h; i:=i+1; q:=q-1;
end;
h:=-h; t:=k;k:=L; L:=t;
until m=0;
f:=not(f); p:=2*p;
until p>=n;
if not(f) then for i:=1 to n do a[i]:=a[i+n] ;
(вывод результата) . for i:=1 to N do begin
write(a[i], ' ');
end;
readin;
end.

```

Рассмотренные два предыдущих примера иллюстрируют большие проблемы сортировки внешних файлов, если в них часты изменения элементов, например, удаления, добавления, корректировки существующих.

В подобных ситуациях эффективными становятся алгоритмы, в которых обрабатываемые элементы представляются в виде структур данных, удобных для поиска и сортировки. В качестве структур данных можно отметить, в частности, линейные списки, очереди, стеки, деревья и т.п. О них было рассказано в предыдущем разделе.

Контрольные вопросы и задания

1. Как в общем случае формулируется задача поиска? сортировки?
2. Почему внутренняя и внешняя сортировки реализуются разными методами?
3. В чем состоят принципы линейного поиска? поиска делением пополам?
4. Какие вы знаете методы внутренней сортировки?
5. Как соотносятся эффективности различных методов сортировки массивов?
6. В чем состоит принцип метода слияния упорядоченных файлов?
7. Разработайте программу упорядочивания списка группы студентов:
 - а) методом прямого включения;
 - б) методом выбора;
 - в) методом обмена.

§ 5. БЕЙСИК КАК ЯЗЫК ОПЕРАЦИОНАЛЬНО-ПРОБЛЕМНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

У языка Бейсик (Basic) весьма своеобразная судьба. Будучи созданным для, так называемых, непрофессиональных программистов, многократно раскритикованный почти каждым пишущим о программировании, он живет уже четверть века и продолжает иметь множество пусть не

поклонников, но пользователей. В своих старших версиях он давно перестал быть столь «простым» как его принято почему-то представлять. Его возможности чрезвычайно велики, о чем можно судить хотя-бы по названию одной из недавно вышедших книг - «Разработка экспертных систем на языке Бейсик». На нем создают программы самой различной предметной ориентации. По-видимому, Бейсик продолжает лидировать по количеству пользователей, и хотя бы поэтому знакомство с ним необходимо.

В данном учебнике нет регулярного, «по-порядку». изложения Бейсика. Для человека, освоившего Паскаль, приведенного ниже в этом параграфе текста достаточно, чтобы составить себе отчетливое представление о Бейсике. Количество же учебников по нему столь велико, что нет смысла приводить их список - достаточно заглянуть в любой библиотечный каталог.

Даже при беглом знакомстве обращает на себя внимание некоторая «недисциплинированность» Бейсика - с точки зрения программиста, привыкшего к структурному языку семейства Паскаля. Бейсик относится к языкам операциональным, рожденным от вечно живого Фортрана, в которых необязательно (хотя и вполне возможно) организовывать строго упорядоченные программные структуры. Это и большой недостаток (особенно при разработке крупных программных комплексов), но иногда и достоинство - например, при разработке относительно небольшой диалоговой программы с регулярным обращением к внешним устройствам, сканированием клавиатуры и т.п.

Еще одна проблема, систематически возникающая при работе с Бейсиком - обилие версий и фактическое отсутствие базовой версии. Оставив обзор до конца данного параграфа, укажем лишь, что многие команды и функции в разных версиях сильно различаются, а иногда, существуя в одной, вовсе отсутствуют в другой. Это следует иметь в виду, если приведенные ниже примеры будут не просто анализироваться, а выполняться на ЭВМ, или по их аналогии будут разрабатываться собственные программы. Справочник по реально используемой версии в таком случае просто необходим.

Тексты приведенных в качестве примеров программ отлажены в широко распространенной версии языка QuickBasic.

5.1. ВВЕДЕНИЕ В БЕЙСИК

Выполнять в среде Бейсика элементарные операции и вычисления, особенно в ранних версиях типа GW-Basic или MSX-Basic, действительно нетрудно. Если компьютер включен и Бейсик загружен, можно смело приступить к работе. Начнем с того, что вы хотите что-то вычислить. Бейсик для этого лучше, чем любой калькулятор. Наберите команду

```
PRINT "Это команда вывода", 5*5
```

и нажмите клавишу <Enter> (слово PRINT может заменить знак ?). Немедленно возникает ответ: 25. Команда PRINT выводит на экран результат вычислений или сообщений, заключенных в кавычки: PRINT "Привет!"- на экране появилось «Привет!» (без кавычек).

При вычислениях необязательно, чтобы операндом было выражение, содержащее только числа- Попробуйте ввести следующие команды (заканчивая каждую строчку нажатием на клавишу <Enter>):

```
a=5 b=4 ?a*b
```

Компьютер немедленно выдаст результат: 20. , . ;

Режим работы, описанный выше, часто так и называют- режим калькулятора (или непосредственный режим).

А теперь каждую из представленных выше трех команд пронумеруем, см. программу 54.

Программа 54

```
10 a=5 20 b=4
```

```
30 ? a*b
```

Обратим внимание, что после ввода этой программы команды не выполнялись, а записывались в память компьютера. Убедиться в этом можно введя команду LIST -текст этой маленькой программы тут же появится на экране.

Если теперь ввести команду RUN, то она запустит программу на исполнение. Таким образом, последовательный набор команд с номерами строк является программой на языке Бейсик. Программировать на языке Бейсик означает научиться составлять определенный набор команд для

решения поставленной вами задачи. Какие имеются команды у Бейсика и как ими пользоваться - рассмотрим ниже. Режим, при котором команды не выполняются непосредственно, а «копятся», называют косвенным. В этом режиме, основном для Бейсика, он и является языком программирования.

Следует помнить, что существует множество версий языка Бейсик и все они имеют особенности. Описать здесь все версии не представляется возможным, да и нет смысла. В каждой из современных версий Бейсика можно выделить общее подмножество, в котором отражены характерные (стандартные) грамматика, синтаксис и семантика языка. В этой связи в последующих описаниях языка рассматривается лишь выделенное авторами подмножество, справедливое для наиболее популярных в настоящее время версий Бейсика: Basic-MSX, Qbasic, Turbo-Basic. Последние версии приобрели популярность благодаря удобному интерфейсу и предоставлению пользователю ряда сервисных возможностей, присущих современным системам программирования.

Контрольные вопросы

1. Чем принципиально различаются прямой и косвенный режимы в Бейсике?
2. Обязательно ли в Бейсике явное описание типов данных?

5.2. БАЗОВЫЕ ОПЕРАТОРЫ

Основные базовые операторы (команды) языка Бейсик определяют ввод и вывод данных, присвоение, изменение порядка выполнения команд и циклические конструкции.

INPUT <список объектов ввода>	- ввод данных;
PRINT < список объектов вывода>	- вывод данных;
LET a= <арифметическое, логическое или символьное выражение> (служебное слово LET можно не писать)	- присвоение;
IF <условие> THEN <оператор1> ELSE <оператор2>	- условный оператор;
GOTO <номер строки>	-безусловный переход;
FOR x= 1 TO n STEP h <оператор> NEXTx	- циклическая конструкция.

Часто используют, так называемый, внутренний ввод данных посредством операторов READ - DATA.

Добавим к этому списку несколько системных команд, с помощью которых программист и пользователь занимаются отладкой и обслуживанием программы:

RUN	- команда запуска программы на выполнение;
LIST	- команда вывода текста программы на экран дисплея;
SAVE	- команда сохранения текста программы в виде файла;
LOAD	- загрузка ранее сохраненной программы из существующего файла.

Этих операторов и команд обычно хватает, чтобы написать и отладить любую вычислительную программу. Ниже мы познакомимся и с другими командами Бейсика.

Как и во многих языках программирования, в Бейсике имеется набор встроенных функций: математических, логических, символьных и др. Можно сформировать собственные функции с помощью описания DEF, например

$$\text{DEF FNA}(x,y,z)=x*x+y*y+z*z$$

Рассмотрим пример программы табуляции функции с целью определения ее максимального значения на заданном отрезке. Суть алгоритма заключается в вычислении значений функции $\text{Sin}(x)$ в 100 точках, определенных на задаваемом отрезке $[a,b]$ с шагом $h=(b-a)/100$ и в выборе среди этих значений максимального.

Программа 55

10 REM максимум функции на отрезке

```

20 INPUT "введите отрезок a,b"; a, b
30 DEF fna (x) = SIN(x)
40 max = fna (a)
50 h = (b - a) / 100
60 FOR x = a TO b STEP h
70 IF max < fna(x) THEN max = fna(x)
80 NEXT x
85 CLS
90 PRINT "максимальное значение функции на отрезке ["; a;
  ", "; b; "]="; max
100 END

```

Здесь в строке 10 - неисполнимый комментарий (который можно организовать и иначе - см. первые строки последующих программ), в строке 85 - команда очистки экрана, в строке 100 - команда «конец программы» (которую часто можно не писать без каких-либо видимых последствий).

Приведем примеры еще нескольких программ на Бейсике, являющихся аналогами программ из §3.

В следующем примере развилка организована с помощью сочетания операторов IF...THEN и GOTO. В современных версиях Бейсика есть конструкция IF...THEN...ELSE; при ее использовании программа 56 станет еще больше похожей на программу 1 (sqr(x) - корень квадратный).

Программа 56 (см. программу 1)

```

10 REM Квадратное уравнение
20 CLS
30 INPUT "введите коэффициенты A,B,C"; a, b, c
40 d = b*b - 4*a*c
50 IF d < 0 THEN GOTO 80
60 x1 = (-b + sqr(d)) / (2 * a): x2 = (-b - sqr(d)) / (2 * a)
70 PRINT "корни уравнения x1="; x1; "x2="; x2
80 GOTO 100
90 PRINT "корней нет"
100 END

```

В следующем примере цикл (с предусловием) организован с помощью операторов IF...THEN и GOTO.

Программа 57 (см. программу 5).

```

10 REM Машины, прибывающие на склад
20 CLS
30 num = 0: sum = 0
40 IF sum >= 100 THEN GOTO 110
50 INPUT "Введите вес груза очередной машины"; w
60 sinn = sum + w
70 IF sum >= 100 THEN GOTO 90
80 num = num + 1: GOTO 90
90 PRINT "груз уже не поместится"
100 GOTO 40
110 PRINT "Количество разгруженных машин =", num
120 END

```

В программе 58 исходная последовательность задается в блоке данных (строка 30), а затем считывается в массив a. Отсутствие в Бейсике логических переменных преодолено с помощью текстовой переменной p. При выдаче результатов используется форматная печать PRINT USING (для дробных чисел форматы могли бы быть, например, ###.## - выдаст результат типа 345.98, ##.##¹¹¹¹ - выдаст результат типа 34.17E-02). В 20-й строке оператор DIM резервирует память под массив (аналог паскалевского описания array, но без явного указания типа элементов).

Программа 58 (см. программу 10).

```

10 REM bubble (пузырек)
20 DIM a(10)
30 DATA 19, 8, 17, 6, 15, 4, 13, 2, 11, 0

```

```

40 CLS
50 PRINT "Исходный массив"
60 FOR i = 1 TO 10 STEP 1
70 READ a(i)
80 PRINT USING ">#>#"; a(i);
90 NEXT i
100 PRINT
110 p$ = "да"
120 FOR i = 10 TO 2 STEP -1
130 IF a(i) >= a(i-1) THEN GOTO 150
140 b = a(i): a(i) = a(i-1): a(i-1) = b: p$ = "нет"
150 NEXT i
160 IF p$ = "нет" GOTO 110
170 PRINT "Упорядоченный массив"
180 FOR i = 1 TO 10
190 PRINT USING "####"; a(i);
200 NEXT i
210 END

```

Контрольные вопросы к заданиям

1. Назовите основные операторы языка Бейсик и охарактеризуйте их назначение.
2. Поэкспериментируйте в приведенных выше программах с оператором форматной печати.

5.3. МУЗЫКАЛЬНЫЕ ВОЗМОЖНОСТИ

В ряд версий Бейсика встроен специальный подязык, который позволяет программировать музыку. Это делается без труда командой **PLAY**. В команде указывается строковое выражение, составленное из символов, кодирующих ноты.

Общий формат команды **PLAY** таков:

PLAY AS, BS, CS,

где **AS, BS, CS** - строки символов, число которых не превышает числа параллельных звуковых каналов в системе. Каждый символ - команда специального языка, задающая ноту, длительность и громкость звучания, другие параметры. Следующие латинские буквы соответствуют основным нотам:

c - «до»,
d - «ре»,
e - «ми»,
f - «фа»,
g - «соль»,
a - «ля»,
b - «си»

Например, команда

PLAY "cdefgab"

осуществляет музыкальное исполнение гаммы (до-ре-ми-фа-соль-ля-си) четвертой октавы с длительностью 1/4 ноты (устанавливаемой, при отсутствии явных указаний, «по умолчанию»).

Нота может звучать на полтона выше (бемоль - знак «минус») или ниже (диез - знак «плюс», либо #). Высота звучания ноты определяется октавой. Октава кодируется символом с номером **On**, помещаемым перед нотой или группой последующих нот. Принято кодировать октавы по порядку с единицы в соответствии с фортепианным рядом, первая октава (01) - контроктава, вторая октава (02) - большая октава, третья октава (03) - малая октава, четвертая октава (04) - первая октава, пятая октава (05) - вторая октава и т.д. Например, команда

PLAY "o3 a a a"

проигрывает три ноты ля-ля-ля в малой октаве.

Длительность звучания ноты измеряется в специальных единицах - одна целая нота (произнесите четко «раз-и, два-и, три-и, четыре-и», тогда время, затраченное на эту фразу, приблизительно соответствует длительности одной ноты). Выделяют длительность в полноты (1/2 ноты), четверть, одну восьмую, одну шестнадцатую и т.д. - вплоть до одна шестьдесят четвертая. Длительность звучания в программе устанавливают командой Lp (p = 1, 2, ... 64), включаемой в символическую последовательность, либо непосредственно в самих командах установки нот. Например,

PLAY"C8D2L16fbe"

означает звучание ноты «до» длительностью в одну восьмую ноты, «ре» - в одну вторую, а ноты фа-си-ми звучат длительностью в одну шестнадцатую ноты. Аналогичную размерность имеет пауза, обозначаемая Rp или Pp.

Уровень громкости может быть установлен командой Vn, предшествующей ноте (n = 0, 1, ... 15); по умолчанию n = 8 - «средняя» громкость.

Ниже приведен пример кодирования музыкального произведения.

Программа 59

```
***** рпезы Шумана *****
10 DATA c8, f2f8, e8, f8, a8, o5c8, f8f2
20 DATA e8, d8, c8, f8, o4g8, a8, b-8, o5d8, o4f8, g8, a8
30 DATA o5c8, o4g2, c4, f2f8, e8, f8, a8, o5c8, a8a4a8
40 DATA g8, f8, d8, c8, f8, o4, g8, a8. b-8, o5, d8, o4, g8, a8, b-8
50 DATA o5d8, o4-d8, e8, f2
60 FOR i = 1 TO 46
70 READ f$
80 PLAY f$
90 NEXT i
```

Использование функции считывания n символов нажатых клавиш INPUTS(n) позволяет использовать клавиатуру как музыкальный инструмент.

Программа 60

```
5 ***** компьютер - простейший музыкальный инструмент *****
10 f$ = INPUT$(1)
20 PLAY f$
30 GOTO 10
```

Контрольные вопросы и задания

1. Составьте программу, исполняющую первые такты полонеза Огинского.
2. Разработайте компьютерный вариант популярной игры «Угадай мелодию».

5.4. ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ

Для получения графических образов необходимо задать графический экран (например, SCREEN 9) и масштаб или окно экрана WINDOW (x1,y1) - (x2,y2). Здесь (x1,y1), (x2,y2) - координаты угловых точек экрана, расположенных на диагонали. Если ограничиться этим, то цвет фона будет черным. Для задания фона другого цвета используется оператор COLOR z1, z2, где z1- цвет текста, z2- цвет фона. Например, зададим графический экран, с размерами по горизонтали и по вертикали, соответственно 32 и 24 единицы, с началом координат в центре экрана. Цвет фона- синий (z2=1), а цвет надписей -желтый (z1=14).

```
10 ***** ИГРНЫЙ ЛИСТ *****
20 SCREEN 9
30 WINDOW (-16,-12)-(16, 12)
```

Изображение точки с координатами (x,y) и заданным цветом осуществляется командой PSET(x,y),z. Здесь z - номер цвета. Точку на бумаге можно изобразить маленьким кружком. Иногда, когда требуется изображение точки покрупнее, буквально рисуют кружок и закрашивают его. В этой связи, изображение точки на экране дисплея можно создавать, выводя образ окружности и закрашивая ее в тот или иной цвет. Для создания окружности используется оператор CIRCLE (x,y),r,z, где x,y - координаты центра окружности, r - радиус окружности, z - номер цвета окружности. Закрашивают окружность оператором PAINT (x,y),zl,z, где zl - номер цвета которым закрашивают, z - номер цвета окружности.

Пример. При запуске программы на экране появятся 4 «точки» в углах квадрата 10*10, окрашенные в зеленый цвет (10), и пятая «точка» красного цвета (12) вдвое большего размера в центре экрана. Их границы окрашены в белый цвет (15).

Программа 61

```
10 '***** Точки *****
20 SCREEN 9
30 WINDOW (-16,-12) - (16,12)
40 COLOR 14, 1
50 '***** изображение точек *****
60 CIRCLE (-5, -5), .25, 15 : PAINT (-5, -5), 10, 15
70 CIRCLE (-5, 5), .25, 15 : PAINT (-5, 5), 10, 15
80 CIRCLE (5, 5), .25, 15 : PAINT (5, 5), 10, 15
90 CIRCLE (5, -5), .25, 15 : PAINT (5, -5), 10, 15
100 CIRCLE (0, 0), .5, 15 : PAINT (0, 0), 12, 15
```

Допустим, что в компьютерной демонстрации надо обратить внимание на какую-либо точку графического образа. В этом случае точку делают мигающей. Для этого меняют с определенной частотой ее цвет.

Программа 62

```
10 '***** ЧИСТЫЙ ЛИСТ *****
20 SCREEN 9
30 WINDOW (-16, -12) - (16, 12)
40 COLOR 14, 1
50 '***** мигающая точка *****
60 FOR I=1 TO 5
70 CIRCLE (0, 0), .5, 15 : PAINT (0, 0), 14, 15 : DELAY .2
80 'FOR k=1 TO 200 : NEXT k
90 CIRCLE (0, 0), .5, 15 : PAINT (0, 0), 12, 15 : DELAY .5
100 'FOR k=1 TO 500 : NEXT k
110 NEXT I
```

Обратим внимание на то, что в этой программе время выдержки красного цвета (12) полсекунды (DELAY .5, либо пустой цикл), а желтого цвета (14) - две десятых секунды. В результате получается красная мигающая точка.

Для акцентирования внимания мигание точки можно сопроводить звуковыми сигналами. Для этого используют оператор BEEP стандартной частоты и длительности. Для издания звукового сигнала заданной частоты и длительности применяют оператор SOUND cl,t, где cl - частота сигнала, t - длительность.

Программа 63

```
10 ***** Клаксон *****
20 SCREEN 9
30 WINDOW (-16, -12) - (16,12)
40 COLOR 14, 1
```

```

110 '***** мигающие точки со звуковым сигналом BEEP *****
120 FOR i=1 TO 5
130 CIRCLE (-5, -5), .25, 15 : PAINT (-5, -5), 6, 15
140 CIRCLE (-5, 5), .25, 15 : PAINT (-5, 5), 6, 15
150 CIRCLE (5, 5), .25, 15 : PAINT (5, 5), 6, 15
160 CIRCLE (5, -5), .25, 15 : PAINT (5, -5), 6, 15
170 DELAY .2 : BEEP
180 CIRCLE (-5, -5), .25, 15 : PAINT (-5, -5), 10, 15
190 CIRCLE (-5, 5), .25, 15 : PAINT (-5, 5), 10, 15
200 CIRCLE (5, 5), .25, 15 : PAINT (5, 5), 10, 15
210 CIRCLE (5, -5), .25, 15 : PAINT (5, -5), 10, 15
220 DELAY .5
230 NEXT i
240 '***** Мигающая точка со звуковым сигналом SOUND *****
250 FOR i=1 TO 5
260 CIRCLE (0, 0), .5, 15 : PAINT (0, 0), 14, 15
270 SOUND 600, .2 : DELAY .2
280 CIRCLE (0, 0), .5, 15 : PAINT (0, 0), 12, 15
290 SOUND 300, .5 : DELAY .5
300 NEXT i

```

В следующем примере демонстрируется работа с окружностями и эллипсами, которые достаточно часто встречаются в графических образах различных объектов. Изображение окружности или дуги окружности выводится тонкими линиями с помощью оператора CIRCLE.

Программа 64

```

'*****Окружности и дуги *****
20 SCREEN 9
30 WINDOW (-16,-12) - (16,12)
40 COLOR 14,1
50 Г***** задание числа PI *****
60 pi=4*atn(1)
   IQ >***** изображение окружности и дуг *****
80 CIRCLE (0, 0), 4, 15
90 CIRCLE (5, 5), 4, 14, 0, pi/2 'дуга в первой четверти
100 CIRCLE (-5, 5), 4, 14, pi/2, pi 'во второй
110 CIRCLE (-5,-5), 4, 14, 0, pi, 3*pi/2 'в третьей
120 CIRCLE (5,-5), 4, 14, 3*pi/2, 2*pi 'в четвертой

```

Для изображения линий используется оператор LINE, который строит линии одной и той же толщины. Однако, при создании графических образов в компьютерных Демонстрациях требуются линии различной толщины. Например, выделяются оси координат на фоне координатной сетки или рисуется график линейной зависимости и т.д. Существует очень быстрый способ получения изображений вертикальных и горизонтальных линий различной толщины. Это достигается выводом на экран узких прямоугольников. Этот прием иллюстрируется на примере построения осей координат:

```
LINE(-16,-1)-(16,1),15,bf LINE(-1,-12)-(.1,12),15,bf
```

Линии произвольной формы и толщины, в том числе и прямые, выводятся на экран дисплея при помощи оператора CIRCLE. Заключение в цикл с достаточно малым шагом, оператор CIRCLE действует аналогично плакатному перу с шириной линии, равной диаметру окружности:

```

FOR x=-4 TO 4 STEP .01
  y = 0.5*x^2
  CIRCLE (x,y),.1,15 NEXT x

```

Если требуется провести очень тонкую линию, то вместо CIRCLE используется оператор PSET (x,y), z, где z - номер цвета, а (x,y) - координаты точки (PSET -точка). Иногда изображения соседних точек соединяют отрезками прямых линий. График функции в этом случае будет представляться ломаной линией.

Программа 65

```
Г ***** тонкие линии и ломаные кривые *****
```

```

10 SCREEN 9
20 WINDOW (-16,-12) - (16,12)
30 COLOR 14,1
40 y***** тонкая линия *****
50 FOR x = -15 TO 15 STEP .01
60 y = 3*sin(x/3)
70 PSET (x, y+7), 14
80 NEXT x
90 r***** ломаная линия - 1 *****
100 FOR x = -15 TO 15 STEP .1
110 y = 3*sin(x/3)
120 PSET (x,y), 11
130 IF x>-15 THEN LINE (x,y) - (x1.y1), 11
140 y1=-y : x1=x
150 NEXT x
160 ***** ломаная линия - 2 *****
170 FOR x = -15 TO 15 STEP .1
180 y = 3*sin(x/3)
190 CIRCLE (x, y-7), .1, 15
200 IF x>-15 THEN LINE (x, y-7) - (x1, y1- 7), 10
210 CIRCLE (x1, y1- 7), .1, 15 : PAINT (x1.y1 - 7), 14, 15
220 y1=y : x1=x
230 NEXT x
340 '***** Оси *****
250 LINE (-16, 7) - (16, 7), 15
260 LINE (-16, 0) - (16, 0), 15
270 LINE (-16, -7) - (16, -7), 15

```

Выше упоминалось о способе закрашивания областей, ограниченных линией одного цвета, с помощью оператора PAINT. Оператор LINE (x1,y1)-(x2,y2),z,bf позволяет получать закрашенные прямоугольники. Существуют специальные приемы закрашивания. Они основаны на повторении оператора, закрашивающего элементарную площадку. Причем, это может производиться либо по определенному закону, либо случайным образом. Ниже приведена программа, иллюстрирующая два способа закрашивания (хотя можно придумать и другие).

Программа 66

```

10 ***** закрашивание *****
20 SCREEN 9
30 WINDOW (-16, -12) - (16, 12)
40 COLOR 14, 1
50 '***** постепенное закрашивание *****
60 LINE (-10.1, -5.1) - (10.1, 5.1), 15, b
70 FOR x = -10 TO 10 STEP .01
80 LINE (x, -5) - (x, 5), 12
90 NEXT x
100 DELAY 2 : ' FOR k=1 to 2000 : NEXT k
110 CLS
120 'А***** гаситель экрана *****
130 FOR i = 1 TO 2000
140 x = .5*INT(RND*64) - 16
150 y = INT(RND*24) - 12
160 LINE (x, y) - (x+.5, y+1), 15, bf
170 NEXT i

```

Следующая программа аналогична программе 31 на Паскале и строит столбчатую диаграмму, наглядно отражающую числовую информацию о населении 6 крупных городов мира: Токио, Гамбурга, Москвы, Бангкока, Мехико и Парижа.

Программа 67 (см. программу 31)

```

Ю ***** столбчатая диаграмма *****
20 SCREEN 9
30 COLOR 14, 1
40 DATA 11500, 2300, 9700, 5100, 12400, 8200

```

```

45 DATA "Токио", "Гамбург", "Москва", "Бангкок", "Мехико", "Париж"
50 LINE (40, 300) - (550, 300): LINE (40, 300) - (40, 20)
60 FOR k = 1 TO 6
70 READ m(k)
80 NEXT k
90 a = m(1)
100 FOR k = 2 TO 6
110 IF m(k) > a THEN a = m(k)
120 NEXT k
130 FOR k = 1 TO 6
140 READ name$(k)
150 NEXT k
160 FOR k = 1 TO 6
170 n = 10 * k + 3
180 LOCATE 23, n: PRINT name$(k)
190 m(k) = 300 - m(k) / a * 200
200 LINE (k * 80, 300) - (80 + k * 80, 300 - m(k)), k + 3, BF
210 NEXT k
220 c$ = HEX$(a)
230 LOCATE 3, 1: PRINT c$
240 c$ = HEX$(0)
250 LOCATE 22, 3: PRINT c$ 260 END

```

Дадим короткие пояснения к использованным в данной программе и не использовавшимся ранее средствам. В представленном примере использован не встречавшийся ранее оператор LOCATE X,Y, останавливающий курсор на позицию x,y координатной сетки дисплея. Сочетание команд LOCATE ... PRINT позволяет выводить на экран, находящийся в графическом режиме, тексты. Функция HEX\$ преобразует числа в соответствующие символы. Отсутствие команды WINDOW приводит к максимально возможному графическому окну.

Контрольные вопросы и задания

1. Перечислите основные операторы для построения элементов изображений и охарактеризуйте их работу.
2. Замените в одном из приведенных выше примеров SCREEN 9 на другой экранный режим (например, 12). Что изменилось в работе программы? Если она перестала исполняться, отладьте ее.
3. Изобразите смешную «рожицу», попеременно моргающую глазами с звуковым эффектом.
4. Постройте орнаменты из геометрических фигур.

5.5. ОБРАБОТКА СИМВОЛЬНОЙ ИНФОРМАЦИИ

Кроме числовых значений переменные в Бейсике могут принимать значения в виде строки символов. Символьные значения (константы) заключаются в кавычки " ".

Имя символьной (строковой) переменной отличается от обычной числовой переменной добавлением справа специального символа, чаще знака доллара \$, например, AS, Fl\$, MaxS. Над символьными переменными допустима операция сложения (склейка строк символов); могут быть использованы функции и процедуры обработки символьных величин.

Начнем с примеров программ, аналогичных приведенным ранее на языке Паскаль.

Программа 68 (см. программу 11)

```

10 ' ***** Количество гласных в тексте *****
20 c$ = "аеиоуыэюяАЕИОУЫЭЮЯ"
30 INPUT "Введите текст"; a$
40 n = 0
50 FOR k = 1 TO LEN(c$)
60 b$ = MID$(c$, k, 1)
70 FOR i = 1 TO LEN(a$)
80 IF b$ = MID$(a$, i, 1) THEN n = n + 1
90 NEXT i

```

```

100 NEXT k
110 PRINT "Количество гласных равно", n
120 END

```

Здесь использована строковая функция MID(a\$,i,j) - вырезка подстроки из строки a\$, где первый аргумент - исходная строка, второй - начальная позиция подстроки, третий - длина подстроки. Функция LEN(a\$) - длина текста.

В следующей программе дополнительно использованы строковые функции LEFT\$(a\$,i) - выделение подстроки длиной i от левого края заданной строки и RIGHT\$(a\$,i) - то же, но с правого края.

Программа 69 (см. программу 12)

```

10 ' ***** Замена в тексте sqr на exp *****
20 INPUT "Введите текст"; a$
30 FOR i = 1 TO LEN(a$) - 2
40 IF MID$(a$, i, 3) = "THEN GOTO 60
50 a$ = LEFT$(a$, i - 1) + "+" RIGHT$(a$, LEN(a$) - i - 2)
60 NEXT i
70 PRINT a$
80 END

```

Приведем еще несколько примеров.

Программа 70

```

10 ' ***** игра клавиатурный тренажер *****
12 DATA "азбука", "год", ".23*45/56", letter, pc-ibm
15 DATA "дом-home", "Красноярск", "Омск", "Пермь", "Воронеж"
20 FOR i = 1 TO 10
30 READ a$(i)
40 NEXT i
60 FOR i = 1 TO 10
70 CLS : PRINT : PRINT TAB(10); a$(i);
100 PRINT : PRINT : PRINT
110 PRINT "введите слово и нажмите <BK>";
120 LOCATE 8, 5 : x = TIMER : INPUT b$ : CLS
130 y = TIMER : PRINT : n = LEN(a$(i))
140 IF a$(i) <> b$ THEN 190
150 PRINT USING "###.##<"; (y - x) / 60;
160 PRINT "сек. на один символ": GOTO 200
170 PRINT : PRINT "слово набрано неверно"
180 FOR m = 1 TO 5000 : NEXT m
190 NEXT i

```

Функция TIMER возвращает текущее время системной среды в секундах. В строке 180 предусмотрен пустой цикл для осуществления временной задержки в исполнении программы.

Использование функций LEFT\$(a\$,n) и MID\$(a\$,n,k) еще раз продемонстрировано в следующем примере. Здесь же использованы математические функции INT(x) - целая часть числа x, RND(x) - случайное число в интервале от 0 до 1. В программе задается массив (оператор DIM a\$(k)) из пяти слов, представленных в списке DATA (30 и 40 строки). Случайным образом выбирается одно из них и предлагается его угадать вводом подходящих символов.

Программа 71

```

5 ' ***** угадай слово *****
10 k = 5
15 DIM a$(k)

30 DATA "вечер", "информатика", "университет"
40 DATA "эхо", "тра-ля-ля"
50 FOR i = 1 TO k
60 READ a$(i)
70 NEXT i

```

```

80 CLS
90 r = INT(k * RND(-TIMER)) + 1
100 n = LEN(a$(r))
110 PRINT "Я задумал слово из "; n; "букв"
120 LOCATE 5, 20: PRINT LEFT$(s$, n) III
130 t = 0
140 DO • 150 LOCATE 8, 12: PRINT "введи букву" III
    160 INPUT h$
170 y = 0
180 FOR i = 1 TO n ] 190 IF h$ = MID$(a$(r), i, 1) THEN | 200 LOCATE 5, 19 +
i: PRINT h$: | 210 t=t+1:y=1 | 220 END IF |
    230 NEXT i
    240 LOCATE 8, 12: PRINT " "
    250 IF y < 1 THEN LOCATE,10, 15: PRINT "нет такой буквы"
    260 FOR m = 1 TO 5000 : NEXT m 270 LOCATE 10, 15: PRINT " " |
280 LOOP WHILE t < n I 290 LOCATE 13, 13: PRINT "угадал, поздравляю!" f

```

Обратим внимание на использованный в данной программе цикл DO-LOOP-WHILE, который допустим в некоторых версиях Бейсика (QBASIC, Turbo-BASIC).

Контрольные вопросы и задания

1. Охарактеризуйте использованные выше строковые функции.

5.6. ПОДПРОГРАММЫ

В Бейсике предусмотрена работа с подпрограммами, если необходимо использовать многократно сходные действия, или для блочно-структурированного построения программы. Для организации подпрограмм используют две команды: GOSUB <номер строки> - перейти на подпрограмму: RETURN - возврат.

Во многих версиях Бейсика предусмотрен переключатель подпрограмм ON k GOSUB 100,200,300. Здесь k может принимать значения 1,2,3, что соответствует обращению к одной из трех подпрограмм, начинающихся с номеров 100, 200, 300.

Начнем с примера, аналогичного одному из тех, которые сопровождали рассказ о процедурах Паскаля выше.

Программа 72 (см. программу 17).

```

10 ' ***** квадратные корни и логарифмы*****
15 CLS / 20 " а » 35: c$ = "-"
    30 GOSUB 1000 40 PRINT : PRINT "Таблица квадратных
корней"
50 GOSUB 1000
    55 PRINT
    60 FOR x = 1 TO 10
        70 PRINT USING "x = ##» sqr(x) = t».##»<"; x; SQR(x) :
80 NEXT x
    90 GOSUB 1000
        100 PRINT : PRINT "Таблица логарифмов" ;
110 GOSUB 1000 120 PRINT
    130 FOR x = 1 TO 5 140 PRINT USING "x = ««» log(x) =
»».»»»"; X; LOG(x)
    150 NEXT x
    :60 c$ = "*" 170 GOSUB 1000
180 END
    1000 FOR i = 1 TO a : PRINT c$ : NEXT i
    1010 RETURN

```

Обратим внимание на существенно иной механизм передачи параметров в подпрограмму на Бейсике, нежели в паскалевскую процедуру. Формирование входных параметров производится с помощью присваиваний до обращения к процедуре. Это гораздо менее удобно, чем использование формальных и фактических параметров в Паскале.

Приведем еще несколько примеров работы с подпрограммами.

Программа 73

```
'***** меню графиков функций *****'  
10 CLS : PRINT : PRINT  
20 PRINT " ГРАФИКИ ФУНКЦИЙ"  
30 PRINT  
40 PRINT ". Синус"  
50 PRINT ". Косинус"  
60 PRINT ". Тангенс"  
70 PRINT : PRINT  
80 INPUT "введи номер"; j  
90 SCREEN 2  
100 LINE (200, 0) - -(200, 200) : LINE (0, 100) - (400, 100)  
110 PSET (0, 0), 4  
120 FOR x = - 4 TO 4 STEP 1/32  
130 ON j GOSUB 500, 600, 700  
140 PSET (200 + 32 * x, 100 - 32 * y)  
150 NEXT x  
160 r$ = INPUT$(1)  
170 IF r$ = "K" OR r$ = "к" THEN END ELSE SCREEN 0: GOTO 10  
500 y = SIN(x) : RETURN  
600 y = COS(x) : RETURN  
700 y = TAN(x) : RETURN
```

Подпрограммы, к которым обращаются по GOSUB, сильно уступают истинным процедурам Паскаля, поскольку у них отсутствуют возможности локализации переменных, нет формальных параметров, которые можно замещать при обращении фактическими. Однако, в более поздних версиях (например в Qbasic) предусмотрены подпрограммные единицы SUB - подпрограммы-процедуры и FUNCTION - подпрограммы-функции. В следующем примере использована подпрограмма решения квадратного уравнения SUB kvur(a,b,c) со списком входных данных. В основной программе можно объявить подпрограмму оператором DECLARE, а вызвать ее с помощью команды CALL.

Программа 74

```
5 ' ***** пример работу с подпрограммой *****  
10 DECLARE SUB kvur (a!, b!, c!)  
20 INPUT "введи a, b, c"; a, b, c 30 CALL kvur (a, b, c)  
40 END SUB kvur (a, b, c)  
1000 d.=b*b-4*a*c  
1010 IF d < 0 THEN PRINT "корней нет": GOTO 1050  
1020 x1 = (- b + SQR(d)) / 2 / a  
1030 x2 = (- b - SQR(d)) / 2 / a  
1040 PRINT "="; x1, "="; x2  
1050 END SUB
```

Контрольные вопросы

1. В чем подпрограммы Бейсика принципиально отличаются от процедур Паскаля?

5.7. РАБОТА С ФАЙЛАМИ

Для осуществления связи с внешними периферийными устройствами и с файловой системой операционной среды в Бейсике предусмотрена работа с файлами. В следующем фрагменте программы показаны команды для открытия внешнего файла с именем "aaa3.dat" на активном устройстве внешней памяти для записи (OPEN-FOR OUTPLT-AS ff). Запись в файл осуществляется командой PRINT или WRITE с указанием номера файла (строка 30). По завершении работы с файлом его закрытие осуществляется оператором CLOSE.

```
5 ' ***** работа с файлами для записи *****  
10 OPEN "aaa3.aaf" FOR OUTPUT AS ^3
```



```

20FORi=1TO10
30 PRINT #3, i:' либо эквивалентная команда WRITE #3, i
40 NEXT i
50 CLOSE #3

```

Аналогичные команды используют при извлечении данных из существующего файла. При этом в команде OPEN указывают служебное слово INPUT, а считывание информации происходит по команде INPUT с указанием номера файла.

```

' * (***** работа с файлами для чтения *****)
60 OPEN "aa3.dat" FOR INPUT AS #2 'для чтения
70 FOR i = 1 TO 10
80 INPUT #2, a(i)
90 PRINT a(i)
100 NEXT i
110 CLOSE #2

```

Добавить данные в существующий файл можно, используя служебное слово **APPEND**.

```

' работа с файлами для продолжения записи в существующий файл
120 OPEN "aa3.dat" FOR APPEND AS #1
130 WRITE #1, 13
140 CLOSE #1

```

Ниже приведен пример двух программ, осуществляющих передачу и прием зашифрованного послания. Текст шифруется заменой символа его порядковым номером в русском алфавите со смещением на 10. Например, код буквы "а" равен 11, буквы "б" - 12 и т.д. Программы вполне аналогичны программам 26 на Паскале, разобранным в параграфе 3, хотя для разнообразия способ шифровки принят несколько иным.

Программа 75 (см. программу 27 sekret)

```

5 ' *****«-** шифровка текста *****
10 INPUT "введи текст"; a$ 23 c$ = "абвгдежзийклмнопрстуфхцчшщъыэюя "
30 x$ = " "
40 FOR i = 1 TO LEN(a$)
50 FOR j = 1 TO LEN(c$)
60 IF MID$(a$, i, 1) - MID$(c$, j, 1) THEN k = j: GOTO 90
70 NEXT j
80 PRINT "таких символов в таблице кодов нет": GOTO 160
90 kk = k + 10
100 x$ = x$ + STR$(kk)
110 NEXT i
120 OPEN "letter.txt" FOR OUTPUT AS #1
130 PRINT #1, x$
140 CLOSE »1
150 PRINT "шифровка - "; x$
160 END

```

Результатом работы программы является зашифрованный текст, записанный в файл с именем letter.txt.

Следующая программа считывает информацию из этого файла и проводит его раскодирование, используя указанный выше ключ.

Программа 76 (см. программу 27 retsek)

```

5 ' *****•** расшифровка текста *****•***«*»
10 OPEN "letter.txt" FOR INPUT AS #1
20 INPUT t1, a$
30 CLS
40 PRINT a$
50 c$ = "абвгдежзийклшюпрстуфхцчшщъыэюя "
60 x$ = " "
70 FOR i = 1 TO LEN(a$) STEP 3
80 k = VAL(MID$(a$, i, 2))
90 k = k - 10
100 x$ = x$ + MID$(c$, k, 1)
110 NEXT i

```

```

120 CLOSE #1
130 PRINT "шифровка - "; x$
140 END

```

5.8. СРЕДСТВА И МЕТОДЫ ОРГАНИЗАЦИИ ДИАЛОГА

В **Бейсике** сильно развиты команды и функции ввода-вывода данных, позволяющие осуществлять диалог (интерактивный интерфейс) с пользователем.

Для осуществления клавиатурного управления программой, помимо уже известных операторов ввода INPUT и вывода PRINT, используют функцию INPUT\$(n), возвращающую значение введенной символьной строки из n символов с ожиданием, функцию INKEYS, принимающую символьное значение нажатой клавиши без ожидания, функции STICK(n), STRIG(O), определяющие коды стрелок или направления джойстика (внешнего управляющего устройства для компьютерных игр и тренажеров).

Часто при разработке компьютерных программ необходимо использовать коды клавиш в ASCII-представлении. Следующая программа печатает код нажатой клавиши и может оказаться полезной при отсутствии под рукой таблицы ASCII кодов.

Программа 77 5 ' ***** коды символов *****

```

10 CLS
20 f$ = INPUT$(1)
30 PRINT "символ="; f$; "код="; ASC(f$)
40 PRINT 50 GOTO 20

```

Полную таблицу кодов символов можно получить и другим способом, см. программу 78.

Программа 78 5 ***** до коду печать символа *****

```

10 FOR i = 1 TO 256
20 PRINT "код="; i, "символ="; CHR$(i)
30 a$ = INPUT$(1)
40 NEXT I

```

Функцию INPUT\$ часто используют для приостановки выполнения программы до нажатия любой клавиши:

```
n r$ = INPUT$(1): IF r$ = "" THEN n
```

В отличие от оператора INPLT f\ нкцпя не ставит знак "?" и не высвечивает вводимые символы. Это свойство удобно использовать для защиты доступа к какой-либо информации. В следующем примере демонстрируется фрагмент проверки «своего» клиента с помощью пароля. Если при вводе четырехсимвольного пароля ("") трижды допущена ошибка, программа приостанавливает работу.

Программа 79 ;

```

5 ***** пароль ***** ' *
7 CLS
8 PRINT "нажми любую клавишу"
9 r$=INPUT$(1) : IF r$="" THEN 9
10 code$ = "1996" : k = 0
20 INPUT "Имя"; a$
30 PRINT "Пароль "
40 p$ = INPUT$(4)
50 IF p$ = code$ THEN GOTO 80
60 k = k + 1 70 IF k < 3 THEN GOTO 30 ELSE PRINT "До свидания" : END
80 . PRINT 90 PRINT a$, "добро пожаловать"
100 END

```

С помощью строковой переменной INKEYS аналогично можно осуществлять временную задержку программы до нажатия любой клавиши:

```
n s$=INKEYS: IF s$="" THEN n
```

Ниже приведен пример назначения некоторых клавиш, управляющих процессом рисования точек (простейший графический редактор). Здесь код 27 (строка 20) означает клавишу Esc.

Программа 80

```
• ^ i ***** простейший графический редактор ***** 5 SCREEN 2
6 x = 100: y = 100: h = 2
7 PSET (x, y)
10 g$ = INKEY$
20 IF g$ = CHR$(27) THEN 80
30 IF g$ = "A" OR g$ = "a " THEN x = x - h
40 IF g$ = "S" OR g$ = "s " THEN x = x + h
50 IF g$ = "Z" OR g$ = "z " THEN y = y + h
60 IF g$ = "W" OR g$ = "w " THEN y = y - h
70 GOTO 7
80 END
```

При разработке компьютерных программ полезно бывает использовать функциональные клавиши в качестве «горячих» (HOT KEY). Назначение команд функциональным клавишам осуществляет оператор KEY. Его работу можно изучить по следующей демонстрационной программе.

Программа 81

```
5 ' ***** назначение функциональных клавиш *****
10 FOR i = 1 TO 10
20 KEY i, "+ STR$(i)
30 NEXT i
40 KEY 30, "
50 KEY 31, "+ CHR$(13)
60 KEY LIST
70 KEY ON
80 r$ = INPUT$(1): IF r$ = "" THEN 80
```

Контрольные задания

1. Составьте программу, подсчитывающую количество слогов в заданном слове.
2. Разработайте собственный компьютерный вариант игры «Поле чудес».
3. Придумайте новые варианты программ шифрования и дешифрования текста.
4. Разработайте клавиатурный тренажер, используя игровые элементы.
5. Разработайте игру «Крестики-нолики».
6. Разработайте компьютерный вариант карточной игры «21».

5.9. ВЕРСИИ БЕЙСИКА

Бейсик характерен необычайным обилием версий, которые иногда настолько отличаются друг от друга, что могли бы считаться разными языками.

Хотя первая официальная публикация по Бейсику создавшей его группы разработчиков Дартмутского колледжа (США) относится к 1967 г., к этому времени уже было создано три «Дартмутских версии»; первая из них относится к 1964 г. После 1967 г. к разработке этого языка подключились крупные фирмы, и были созданы версии для систем с разделением времени, для сетевых информационных систем, для мощных ЭВМ третьего поколения (включая отечественную версию «Бейсик ЕС»), для малых ЭВМ.

Большой толчок развитию Бейсика дало появление персональных компьютеров. На какое-то время Бейсик с его естественной ориентацией на трансляторы интерпретирующего типа оказался доминирующим для «персоналок» языком программирования. Бейсик оказался пригоден для первых учебных и бытовых ЭВМ с малым объемом оперативной памяти и невысоким уровнем системного программного обеспечения. Зачастую Бейсик-интерпретатор выполнял и функции операционной системы (например, бывшая популярной в России в конце 80-х - начале 90-х годов версия MSX-Бейсик учебных компьютеров «Ямаха»).

Бейсик начала 60-х годов был прост и примитивен. Он насчитывал всего порядка двух десятков команд. Несмотря на простоту работы с Бейсиком, интерфейс пользователя оставлял желать лучшего. Вся информация (текст программы, «мусор», результаты работы, сообщения) раз-

мещались на скроллирующем экране. В силу несложной процедуры разработки транслятора-интерпретатора Бейсика каждый разработчик персональных компьютеров встраивал (обычно в ПЗУ) свой вариант Бейсика.

Бурное развитие микропроцессорной техники в 70-х, затем в 80-х годах и связанные с этим проблемы совместимости заставили разработчиков задуматься о стандарте Бейсика для персональных компьютеров. Аппаратное и программное развитие позволило существенно расширить возможности языка. Бейсик стал системой программирования со всеми сервисными утилитами и инструментарием программиста. Наибольшей популярностью для школьного и педвузовского образования приобрел Бейсик-MSX. Для большинства учебных компьютеров (Ямаха, Корвет, УКНЦ) были разработаны трансляторы версии Бейсик-MSX. Описание и практика работы с этим языком достаточно полно раскрыты в существующих популярных учебниках для вузов и школ.

Бейсик последних версий, в основном, отличается интерфейсом. Например, основное меню Турбо-Бейсика расположено в верхней строке экрана и представляет систему ниспадающих подменю, команды которых позволяют выполнять все необходимые действия по составлению, отладке и выполнению программ. В Турбо-Бейсике существуют несколько экранных режимов. Основной исходный экран представляет поле редактирования текста программы. Встроенный текстовый редактор выполняет две функции - редактирования текста и синхронной интерпретации строки. В этой связи строки нумеровать необязательно. Результаты работы программы, комментарии и сообщения интерпретатора представляются в текстовом экране (по умолчанию режим SCREEN 0). При работе с графикой изображения строятся в графическом экране, который имеет несколько режимов (режимы SCREEN n, n=2,3,...). Аналогичные возможности предусмотрены и в QBasic. Следует отметить, что в последних версиях Бейсика предусмотрены управляющие структуры. Введены операторы выбора SELECT CASE, циклические конструкции с пост- и предусловием WHILE-WEND, DO-LOOP-WHILE, DO-WHILE-LOOP, развит аппарат подпрограмм (SUB-CALL).

Особый интерес представляет система помощи - help (подсказок), имеющая контекстную структуру с возможностью копирования примеров из текста подсказок, что позволяет осваивать работу с языком программирования практически самостоятельно.

5.10. БЕЙСИК И ПАСКАЛЬ

Сопоставим эти популярные языки - не в деталях, а в целом. В Бейсике отсутствуют:

- ряд структур данных (множества, записи, перечисляемые и интервальные типы);
- ссылочные типы и динамические переменные (хотя в старших версиях возможно динамическое описание массивов);
- в большинстве версий - процедуры и функции (их слабыми аналогами являются подпрограммы);
- модули (и, следовательно, возможности организации больших внешних библиотек).

По работе с графикой, возможностям обработки строк (текстов) Бейсик не уступает Паскалю, а по возможностям организации диалога, пожалуй, даже превосходит.

Одна из неприятных особенностей Бейсика для тех, кто привык к структурной записи программного кода, состоит в том, что Бейсик провоцирует программиста к неструктурности. Впрочем, это вопрос навыков и привычек. По большому счету, однако, язык, в котором нет настоящих процедур и средств создания внешних библиотек, не может быть языком структурного программирования.

Усложненные версии Бейсика, в которых фигурируют истинные процедуры и другие средства, заимствованные из более мощных языков, производят неоднозначное впечатление, так как теряется одно из главных достоинств языка - простота (лучше уж тогда пользоваться во многих отношениях превосходящим Бейсик языком операционального программирования Фортран-77). При написании же коротких простых программ Бейсик реально не уступает Паскалю (и многие специалисты утверждают, что процедура программирования на Бейсике проще, хотя это спорно). Поскольку современные версии Бейсик-систем используют не интерпретаторы, а компиляторы, эффективность программ не ниже, чем аналогичных на Паскале.

§ 6. ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ СИ

6.1. ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКА И ПРИМЕР ПРОГРАММЫ НА СИ

Язык программирования Си - это универсальный язык с богатым набором операторов и компактным способом записи выражений. Благодаря гибкости, выразительности и компактности своих конструкций Си завоевал наибольшую популярность в среде профессиональных программистов и широко используется при разработке системных и прикладных программ.

Язык Си представляет собой удачный компромисс между желанием располагать теми возможностями, которые обычно предоставляют программисту столь понятные и удобные языки высокого уровня, и стремлением эффективно использовать особенности компьютера. Кроме набора средств, присущих современным языкам программирования высокого уровня (структурность, модульность, определяемые типы данных) в него включены средства для программирования «почти» на уровне ассемблера (использование указателей, побитовые операции, операции сдвига). Большой набор операторов позволяет писать компактные и эффективные программы. Однако, такие мощные средства требуют от программиста осторожности, аккуратности и хорошего знания языка со всеми его преимуществами и недостатками. В отличие от языков программирования типа Паскаль, требующих жесткой дисциплины программирования, ограничивающих свободу программиста, содействующих устранению многих ошибок в программах еще на стадии их трансляции, язык Си предоставляет программисту наибольшую свободу. Однако, ответственность за корректность программ при этом полностью ложится на программиста.

Си был создан Денисом Ритчи как инструмент для разработки операционной системы UNIX и реализован в рамках этой операционной системы. Название языка имеет случайное происхождение: «С» - третья буква английского алфавита. Это наименование говорит о чувстве юмора у создателей языка - его предшественником был язык В («В» - вторая буква английского алфавита).

В настоящее время имеется большое количество систем программирования на Си для разных типов компьютеров. Разработано много библиотек модулей, инструментальных средств разработки и отладки, облегчающих создание новых программ. Программы на Си обладают высокой мобильностью, без каких-либо изменений они переносятся, транслируются и выполняются на машинах различных типов.

Язык Си - компилирующего типа. Текст программы на Си, подготовленный с помощью текстового редактора, для получения объектного модуля обрабатывается компилятором, причем компиляция выполняется в два прохода. При первом проходе (претрансляции) обрабатываются строки-директивы, начинающиеся со знака #, при втором - транслируется текст программы и создается объектный (машинный) код. Для получения загрузочного (исполняемого) модуля теперь необходимо отредактировать внешние связи объектного модуля - подсоединить к нему соответствующие библиотечные модули.

Рассмотрим простую программу на языке Си. Такой пример позволит нам выявить некоторые основные черты любой программы, написанной на языке Си.

Программа 82

```
#include<stdio.h>
main() /* Простая программа */ <
    int num;
    num =1, '
    printf ("Это моя 1-я программа на языке Си.\n",num);)
```

Результат выполнения программы:

Это моя 1-я программа на языке Си.

Поясним строки этой программы.

#include<stdio.h> - подключение файла stdio.h, который является частью пакета, имеющегося в любом компиляторе языка Си и описывающего функции ввода-вывода (например, средства взаимодействия программы с терминалом). В качестве имени файла используется аббревиатура английских слов: SStandard Input/Output header - стандартный заголовок ввода-вывода. Данная строка не является оператором языка Си. Символ # оказывает, что она должна быть обработана «препроцессором» компилятора. Препроцессор осуществляет некоторую предварительную обработку текста программы перед началом компиляции.

Main() - имя функции (в переводе с английского main - «главная»). Любая программа, написанная на языке Си, состоит из одной или более «функций», являющихся модулями, из которых

она собирается. Данная программа состоит из одной функции `main`. Круглые скобки указывают именно на то, что `main()` - имя функции. Программа, написанная на языке Си всегда начинает выполняться с функции, называемой `main()`. Программист имеет возможность выбирать имена для всех используемых им функций, кроме `main`. В круглых скобках в общем случае содержится информация (параметры), передаваемые этой функции. В нашем примере параметры не требуются и, следовательно, в скобках ничего не содержится.

`/* Простая программа */`-комментарий. Использование комментариев облегчает процесс понимания программы любым программистом. Большим удобством при написании комментариев является возможность располагать их в той же строке, что и операции, которые они объясняют. Длинный комментарий может помещаться в отдельной строке или даже занимать несколько строк. Все, что находится между символом, указывающим начало комментария `/*`, и символом, указывающим его

конец `*/`, игнорируется компилятором, поскольку он не в состоянии интерпретировать язык, отличный от Си.

`{ }` - фигурные скобки отмечают начало и конец тела функции. Фигурные скобки применяются также для того, чтобы объединить несколько операторов программы в сегмент или «блок».

`int num;` - оператор описания. С помощью такого оператора мы объявляем, что будем использовать в программе переменную `num`, которая принимает целые (`int`) значения.

Точка с запятой в конце строки является частью оператора языка Си, а не разделителем операторов, как в Паскале.

Слово `int` служит «ключевым словом», определяющим один из основных типов данных языка Си. Ключевыми словами называются специальные зарезервированные слова, используемые для построения фраз языка.

В языке Си все переменные должны быть объявлены. Это означает, что, во-первых, в начале программы необходимо привести список всех используемых переменных, а во-вторых, необходимо указать тип каждой из них. Вообще, объявление переменных считается «хорошим стилем» программирования.

`num=1;` - оператор присваивания. Служит для присваивания переменной `num` значения 1.

`printf("это моя %d-я программа на языке Си.\n",num);` - вызов функции `printf()` форматного вывода на печать. Этот оператор выводит на печать значение переменной `num` в формате, содержащемся в строке в кавычках (в данном случае печатается приведенная в кавычках фраза. Символы `%d` указывают компилятору, где в выводимой строке и в какой форме необходимо вставить значение переменной `num`. Символ `%` сигнализирует программе, что, начиная с этой позиции, необходимо вывести значение переменной, а `d` указывает, что число надо печатать в десятичном формате.

Символы `\n` не появляются на экране. Эти символы служат директивой начать новую строку на устройстве вывода. Комбинация символов `\n` на самом деле представляет собой один символ, называемый «новая строка». Для этого символу (`\n`) не существует соответствующей клавиши на клавиатуре. Символ «новая строка» служит примером «управляющей последовательности», которую невозможно ввести с клавиатуры.

В общем случае обращение к этой функции имеет вид:

`printf(<формат>,<выражение1>,<выражение2>,...,<выражениеN>);`

где `<выражение1>`, `<выражение2>`, ..., `<выражениеN>`» - произвольные выражения, результаты которых надо вывести.

Управляющая строка «формат» содержит объекты двух типов: обычные символы, которые просто копируются в выходной поток (печатаются), и спецификации преобразования значений из внутреннего машинного представления в текстовое для вывода на печатающем устройстве.

Каждая спецификация преобразования начинается с символа `%` и заканчивается символом преобразования. Между `%` и символом преобразования могут (не обязательно) находиться

1) знак минус, указывающий, что преобразуемый параметр должен быть выровнен влево в своем поле;

2) целое число, задающее минимальный размер поля: преобразованный параметр будет напечатан в поле-минимум оказанного размера; если в преобразованном параметре символов меньше, чем размещается в указанном поле, то слева будут добавлены пробелы (или справа, если указано выравнивание влево);

3) строка цифр с начальным нулем - лишние позиции поля заполняются нулями, а не пробелами:

4) точка, отделяющая размер поля от последующей строки цифр (только для преобразования строк %s) - строка цифр указывает максимальное число символов, выводимых в одной строке.

Символы преобразования:

d - десятичное число со знаком;

u - десятичное число без знака;

O - восьмеричное число без знака (и без ведущего 0);

x - шестнадцатеричное число без знака (и без ведущего 0);

s - символьная строка;

c - одиночный символ;

f- действительное число в представлении с фиксированной точкой;

e-действительное число в экспоненциальном представлении;

g - наиболее короткое представление действительного числа; и др.

Функцией ввода, аналогичной функции вывода printf(), является scanf() - стандартная функция форматного ввода.

Обращение к этой функции имеет вид

scanf(<формат>,<&имя1>,<&имя2>,...,<&имяN>);

где <имя1>, <имя2>, ..., <имяN> - имена переменных, значения которых надо ввести. Наличие символа "&" перед каждым именем обязательно (кроме переменных строкового типа), его смысл будет пояснен ниже.

При обращении к функции scanf выполнение программы приостанавливается, ожидается ввод значений указанных переменных, после чего работа программы продолжается.

В качестве спецификаций в формате можно использовать те же символы, что и в функции printf(). Спецификации формата должны соответствовать количеству и типу вводимых переменных. В управляющей строке функции scanf нет промежуточных управляющих символов между % и символом преобразования, за исключением "*". Если присутствует символ *, входное поле просто пропускается и никакого присваивания не производится. Обычные символы (кроме °) сопоставляются с очередными (отличными от символов промежутков) символами входного потока, и в случае отличия дальнейший ввод игнорируется.

Программа 83

```
#include<stdio.h>
main()
(
    int data,month,year; char name[15],town[15];
    printf("Как вас зовут? "); scanf("%s",name); printf("Укажите дату, месяц и
                                                год вашего рождения.\nДата:");
    scanf("%d",&data) ;
    printf ("Месяц (числом) :") ; scanf ("%d", &month) ;
    printf("Год:"); scanf("%d",&year);
    printf("А в каком городе? "); scanf("%s",town);
    printf("Вот мы и познакомились...\n");
    printf("Вас зовут %s ",name); printf("Вы родились в городе %s
        (%d.%d.%d)",town,data, month,year);)
```

Результат работы программы:

Как вас зовут? Иван

Укажите дату, месяц и год вашего рождения.

Дата : 23

Месяц (номером): 02

Год: 1054

А в каком городе? Новгород

Вот мы и познакомились...

Вас зовут Иван

Вы родились в городе Новгород

(23.02.1054)

6.2. ЭЛЕМЕНТЫ СИ: АЛФАВИТ, ИДЕНТИФИКАТОРЫ, ЛИТЕРАЛЫ, СЛУЖЕБНЫЕ

СЛОВА

Перечислим основные символы языка Си, образующие его **алфавит**:

1) строчные латинские буквы

abcdefghijklmnopqrstuvwxyz

2) прописные латинские буквы

ABCDEFGHIJKLMNOPQRSTUVWXYZ

3) арабские цифры

0123456789

4) специальные символы

*	звездочка
_	подчеркивание
+	плюс
(	круглая скобка левая
-	минус
)	круглая скобка правая
/	дробная черта
<	меньше
\	обратная дробная черта
>	больше
%	процент
^	стрелка вверх
!	восклицательный знак
[	квадратная скобка левая
=	знак равенства] квадратная скобка правая
?	вопросительный знак
#	номер
:	двоеточие
{	фигурная скобка левая
;	точка с запятой
}	фигурная скобка правая
&	коммерческое "и"(амперсанд)
	вертикальная черта
'	апостроф
~	черта сверху (тильда)
.	точка
"	кавычки
,	запятая пробел

5) *управляющие символы*

\t	горизонтальная табуляция
\n	перевод строки и возврат каретки
\r	возврат каретки
\f	перевод страницы
\b	возврат на шаг (на один символ)

Множество основных символов расширено буквами русского алфавита (строчными и прописными). Они могут быть использованы только в комментариях, символьных константах и строках.

Лексемами называют последовательности символов языка (идентификаторы, служебные слова, константы, строки, составные знаки операций, разделители). Лексемы разделяются пробелами и другими неграфическими символами языка.

Идентификатор - это последовательность латинских букв, цифр и символа "_", начинающаяся с буквы или символа "_".

Прописные и строчные латинские буквы считаются различными! Например, у и Y - это разные имена. Рекомендуется в именах переменных использовать только строчные буквы.

Примеры правильных идентификаторов:

schetchik get_line a 12 Parami _ab

Примеры неправильных идентификаторов:

%ab 12abc -х вася

Литералы - это неизменяемые объекты языка (константы). Литерал может быть числовым, символьным или строковым. Числовые литералы могут быть десятичными (целыми и вещественными, простыми и длинными), восьмеричными, шестнадцатичными.

Примеры.

/* Целые десятичные литералы */

57 32000001 /* длинный*/ 2e3 5E3

/* Вещественные десятичные литералы */

0.00 5.37.1e-3 6.34E-2 .21e+56

Лидирующий нуль (0) указывает на числовой восьмеричный литерал:

030 /* Десятичное 24 */

040 /* Десятичное 32 - символ пробел */

Лидирующий 0x указывает на числовой шестнадцатичный литерал:

0x22 /* Десятичное 34 - символ "*/

0x6C /* Десятичное 108 - символ i */

Символьный литерал - это один символ, заключенный в одинарные кавычки:

-'c'"q' -

"\007" /* Звонок, восьмеричный код после \ */

"\x0a" /* Перевод на новую строку, шестнадцатичный код после \x */

Последовательность символов, заключенных в двойные кавычки, называется строковым литералом. Примеры:

"STRING\n"

"" /* Строчный литерал состоит из одного символа "\0" */

"Очень,\"

"очень,\"

"очень длинный строковый литерал!"

Следующие зарезервированные служебные (ключевые) слова языка запрещено использовать в качестве идентификаторов.

auto	- автоматический;
default	- по умолчанию;
break	-завершить;
do	-выполнить;
case	-вариант;
double	-двойной точности;
char	-символьный;
else	-иначе;
continue	- продолжить;
entry	- вход;
extern	-внешний;
short	-короткий;
for	- для;
sizeof	- размер;
float	-плавающее;
static	-статический;
goto	- перейти;
struct	- структура;
if	— если;
switch	- переключатель;
int	- целое; '
typedef	- определение типа;

long	-длинное;
union	-объединение;
register	-регистровый;
unsigned	-без знака;
return	- возврат;
while	- пока.

6.3. ТИПЫ ДАННЫХ И ОПЕРАЦИИ В ЯЗЫКЕ СИ. ВЫРАЖЕНИЯ

Типы данных. Программа на процедурных языках, к которым относится Си, представляет собой описание операций над величинами различных типов. Тип определяет множество значений, которые может принимать величина, и множество операций, в которых она может участвовать.

В языке Си типы связаны с именами (идентификаторами) величин, т. е. с переменными. С переменной в языке Си связывается ячейка памяти. Тип переменной задает размер ячейки, способ кодирования ее содержимого, допустимые преобразования над значением данной переменной. Все переменные должны быть описаны до их использования. Каждая переменная должна быть описана только один раз.

Описание состоит из спецификатора типа и следующего за ним списка переменных. Переменные в списке разделяются запятыми. В конце описания ставится точка с запятой.

Примеры описаний:

```
char a,b; /*      Переменные a и b имеют тип
char */ int x; /*  Переменная x - типа int
*/ char sym; /*   Описаны переменные sym типа char;
*/ int count,num; /* num и count типа int */
```

Переменным могут быть присвоены начальные значения внутри их описаний. Если за именем переменной следует знак равенства и константа, то эта константа служит в качестве инициализатора.

Примеры: char backch = '\0';

int i = 0;

Рассмотрим основные типы в языке Си.

int - целый ("integer"). Значения этого типа - целые числа из некоторого ограниченного диапазона (обычно от- 32768 до 32767). Диапазон определяется размером ячейки для типа и зависит от конкретного компьютера. Кроме того, имеются служебные слова, которые можно использовать с типом int: short int («short integer» - «короткое целое»), unsigned int («unsigned integer» - «целое без знака»), long int («длинное целое»), которые сокращают или, наоборот, расширяют диапазон представления чисел.

char - символьный («character»). Допустимое значение для этого типа — один символ (не путать с текстом!). Символ записывается в апострофах.

Примеры: 'x"2"?'

В памяти компьютера символ занимает один байт. Фактически хранится не символ, а число - код символа (от 0 до 255). В специальных таблицах кодировки указываются все допустимые символы и соответствующие им коды.

В языке Си разрешается использовать тип char как числовой, т. е. производить операции с кодом символа, применяя при этом спецификатор целого типа в скобках - (int).

float - вещественный (с плавающей точкой). Значения этого типа - числа, но, в отличие от char и int, не обязательно целые.

Примеры:

12.87 -316.12 -3.345e5 12.345e-15

double - вещественные числа двойной точности. Этот тип аналогичен типу float, но имеет значительно больший диапазон значений (например, для системы программирования Borland-C от 1.7E-308 до 1.7E+308 вместо диапазона от 3.4E-38 до 3.4E+38 для типа float). Однако увеличение диапазона и точности представления чисел ведет к снижению скорости выполнения программ и неэкономному использованию оперативной памяти компьютера.

Обратите внимание на отсутствие в этом списке строкового типа. В языке Си нет специального типа, который можно было бы использовать для описания строк. Вместо этого строки пред-

ставляются в виде массива элементов типа `char`. Это означает, что символы в строке будут располагаться в соседних ячейках памяти.

Необходимо отметить, что последним элементом массива является символ `\0`. Это «нуль-символ», и в языке Си он используется для того, чтобы отмечать конец строки. Нуль-символ не цифра 0; он не выводится на печать и в таблице кодов ASCII имеет номер 0. Наличие нуль-символа означает, что количество ячеек массива должно быть, по крайней мере, на одну больше, чем число символов, которые необходимо размещать в памяти.

Приведем пример использования строк.

Программа 84

```
# include<stdio.h> main()
{
    char string[31] ;
    scanf("%s",string) ;
    printf("%s",string);
}
```

В этом примере описан массив из 31 ячейки памяти, в 30 из которых можно поместить один элемент типа `char`. Он вводится при вызове функции `scanf("%s",string)`; "&" отсутствует при указании массива символов.

Указатели. Указатель - некоторое символическое представление адреса ячейки памяти, отведенной для переменной.

Например, `&name` - указатель на переменную `name`;

Здесь `&` - операция получения адреса. Фактический адрес - это число, а символическое представление адреса `&name` является константой типа «указатель».

В языке Си имеются и переменные типа указатель. Точно так же, как значением переменной типа `char` является символ, а значением переменной типа `int` - целое число, значением переменной типа указатель служит адрес некоторой величины.

Если мы дадим указателю имя `ptr`, то сможем написать такой оператор:

```
ptr = &name; /* присваивает адрес name переменной ptr */
```

Мы говорим в этом случае, что `ptr` «указатель на» `name`. Различие между двумя формами записи: `ptr` и `&name` - в том, что `ptr` - это переменная, в то время как `&name` - константа. В случае необходимости можно сделать так, чтобы переменная `ptr` указывала на какой-нибудь другой объект:

```
ptr = &bah; /* ptr указывает на bah, а не на name */
```

Теперь значением переменной `ptr` является адрес переменной `bah`. Предположим, мы знаем, что в переменной `ptr` содержится ссылка на переменную `bah`. Тогда для доступа к значению этой переменной можно воспользоваться операцией «косвенной адресации» * :

`val = *ptr;` /* определение значения, на которое указывает `ptr` */ Последние два оператора, взятые вместе, эквивалентны следующему:

```
val = bah;
```

Итак, когда за знаком `&` следует имя переменной, результатом операции является адрес указанной переменной; `&nurse` дает адрес переменной `nurse`; когда за знаком `*` следует указатель на переменную, результатом операции является величина, помещенная в ячейку памяти с указанным адресом.

Пример: `nurse = 22;`

```
ptr = &nurse; /* указатель на nurse */
```

```
val = *ptr;
```

Результат-присваивание значения 22 переменной `val`.

Недостаточно сказать, что некоторая переменная является указателем. Кроме этого необходимо сообщить, на переменную какого типа ссылается данный указатель. Причина заключается в том, что переменные разных типов занимают различное число ячеек памяти, в то время как для некоторых операций, связанных с указателями, требуется знать объем отведенной памяти.

Примеры правильного описания указателей: `int *pi;` `char *pc;`

Спецификация типа задает тип переменной, на которую ссылается указатель, а символ `*` определяет саму переменную как указатель. Описание вида `int *pi;` говорит, что `pi` - это указатель и что `*pi` - величина типа `int`.

В языке Си предусмотрена возможность определения имен типов данных. Любому типу данных с помощью определения typedef можно присвоить имя и использовать это имя в дальнейшем при описании объектов.

Формат: typedef <старый тип> <новый тип> *Пример:* typedef long LARGE; /* определяется тип large, эквивалентный типу long */

Имена производного типа рекомендуется записывать прописными буквами, чтобы они выделялись в тексте программы.

Определение typedef не вводит каких-либо новых типов, а только добавляет новое имя для уже существующего типа. Описанные таким способом переменные обладают точно теми же свойствами, что и переменные, описанные явно. Переименование типов используется для введения осмысленных или сокращенных имен типов, что повышает понятность программ, и для улучшения переносимости программ (имена одного типа данных могут различаться на разных компьютерах).

Операции. Язык Си отличается большим разнообразием операций (более 40). Здесь мы рассмотрим лишь основные из них, табл. 3.3.

Арифметические операции. К ним относят

- сложение(+),
- вычитание (бинарное) (-),
- умножение (*),
- деление (/),
- остаток от деления нацело (%),
- вычитание (унарное) (-) .

В языке Си принято правило: если делимое и делитель имеют тип int, то деление производится нацело, т.е. дробная часть результата отбрасывается.

Как обычно, в выражениях операции умножения, деления и нахождения остатка выполняются раньше сложения и вычитания. Для изменения порядка действий используют скобки.

Программа 85

```
#include<stdio.h>
main()
(
    int s;
    5 = -3 + 4 * 5 - 6; printf("%d\n",s);
    s = -3 + 4%5 - 6; printf("%d\n",s);
    s = -3 * 4% - 6/5; printf("%d\n",s);
    s= (7 + 6)%5/2; printf("%d\n",s);
}
```

Результат выполнения программы: 11 1 0 1

Таблица 3.3 Старшинство и порядок выполнения операций

Приоритет	Операция	Название	Порядок выполнения
Высший	()	Круглые скобки	Слева направо
	[]	Квадратные скобки	Слева направо
	++	Увеличение	Справа налево
	--	Уменьшение	Справа налево
	(тип)	Приведение	Справа налево
1	*	Содержимое	Справа налево
	&	Адрес	Справа налево
	-	Унарный минус	Справа налево
	!	Логическое «НЕ»	Справа налево
	/~	Инверсия битов	Справа налево
	sizeof	Размер объекта	Справа налево
	*	Умножение	Слева направо

2	\ %	Деление Остаток	Слева направо Слева направо
3	+ -	Сложение Вычитание	Слева направо Слева направо
4	» « >	Сдвиг вправо Сдвиг влево Больше	Слева направо Слева направо Слева направо
5	>= < <=	Больше или равно Меньше Меньше или равно	Слева направо Слева направо Слева направо
6	= = !=	Равно Не равно	Слева направо Слева направо
7	&	Битовое «И»	Слева направо
8	~	Битовое исключающее «ИЛИ»	Слева направо
9		Битовое «ИЛИ»	Слева направо
10	&&	Логическое «И»	Слева направо
11	 = +=	Логическое «ИЛИ» Операция присваивания Справа налево	Слева направо Справа налево
12	- = *= /=	Специальная форма опе- рации присваивания	Справа налево Справа налево Справа налево
	%=		Справа налево

В программировании часто встречаются ситуации, когда надо увеличить или уменьшить значение некоторой переменной на 1. Для этого обычно выполняются оператор присваивания вида: $s = s + 1$;

В языке Си для этих действий существуют специальные операции:

- увеличение (+ +),
- уменьшение (--).

Следующие записи на языке Си являются эквивалентными:

$i = i + 1$ и $i++$; $j = j - 1$ и $j--$;

Символы "++" или "--" записывается после имени переменной или перед ним.

Пример:

```

s + +; /* s      увеличить на единицу
*/ t - -; /* t    уменьшить на единицу
*/ ++ a; /* a     увеличить на единицу
*/ --b; /* b      уменьшить на единицу */

```

Как и обычные присваивания, увеличение и уменьшение можно использовать в выражениях. При этом существенно, с какой стороны от имени стоит знак "+ +" или "- -". Если знак стоит перед переменной (в этом случае говорят о префиксной форме операции), то сначала выполняется увеличение (уменьшение) значения переменной, а лишь затем полученный результат используется в выражении. Если же знак стоит после переменной (постфиксная форма операции), то в выражении используется старое значение переменной, которое затем изменяется.

Пример:

```

int i, j, s;
i = j = 2; /* i и j получают значение 2 */

```

```
s = (i++) + (++j);
```

После выполнения этих действий переменные имеют такие значения:

```
i=3;j=3;s=5.
```

Операции увеличения ++ и уменьшения - - можно применять только к переменным, выражения типа s=(i+j)++ являются незаконными. Кроме того, не рекомендуется:

1) применять операции увеличения или уменьшения к переменной, присутствующей в более чем одном аргументе функции,

2) применять операции увеличения или уменьшения к переменной, которая входит в выражение более одного раза.

Операции отношения и логические операции

Больше или равно	>=
Больше	>
Меньше или равно	<=
Меньше	<
Равно	==
Неравно	!=
Логическое «и»	&&
Логическое «или»	
Отрицание «не»	!

Логическое значение «ложь» представляется целым нулевым значением, а значение «истина» представляется любым ненулевым значением

Выражения, связанные логическими операциями && и ||, вычисляются слева направо, причем вычисление значения выражения прекращается сразу же, как только становится ясно, будет ли результат истинен или ложен.

Старшинство операции && выше, чем у операции ||.

Программа 86

```
#include<stdio.h>
main()
(
    int x, y, z;
    x=1; y=1; z=0; x=x&&y||z; printf("%d\n",x);
    x=x|| !y&&z; printf("%d\n",x) ;
    x=y=1; z=x++-1; printf("%d\n",x);printf("%d\n",z) ;
    z+=-x++ + ++y; printf("%d\n",x) ; printf("%d\n",z);
    z=x/++x; printf("%d\n",x); printf("%d\n",z) ;
}
```

Результат выполнения программы: 1 1 2 0 3 0 4 1

Битовые операции

Битовое «и»	&
Битовое «или»	
Битовое исключающее «или»	~
Сдвиг влево	<<
Сдвиг вправо	>>
Инверсия битов (унарная операция)	\~

Программа 87

```
#include<stdio.h>
main()
```

```
(
    int y, x, z, k;
    x=03; y=02; z=01; k=x|y&z; printf("%d\n",k) ;
    k=x|y&~z; printf("%d\n",k) ;
    k=x^y&~z; printf("%d\n",k) ;
    k=x&y&&z; printf("%d\n",k) ;
    x=1; y=-1;
    k=!x|x; printf("%d\n",k) ;
    k=-x|x; printf("%d\n",k) ;
    k=x^x; printf("%d\n",k) ;
    x<<=3; printf("%d\n",x);
    y<<=3; printf("%d\n",y);
    y>>=3; printf("%d\n",y);
}
```

После выполнения программы получаем следующие результаты:

3 3 1 1 1 -1 0 8 -8 8 1 9 1

Выражения. Конструкции, включающие константы (литералы), переменные, знаки операции, скобки для управления порядком выполнения операций, обращения к функциям, называются выражениями.

Если в выражениях встречаются операнды различных типов, то они преобразуются к общему типу в соответствии с определенными правилами:

- переменные типа char интерпретируются как целые без знака (unsigned);
- переменные типа short автоматически преобразуются в int; если один из операндов имеет тип unsigned, то другой (другие) также преобразуются к типу unsigned и результат имеет тип unsigned;
- если один из операндов имеет тип int, то другой (другие) также преобразуются к типу int и результат имеет тип int;
- если один из операндов имеет тип char, то другой (другие) также преобразуются к типу char и результат имеет тип char;
- во время операции присваивания значение правой части преобразуется к типу левой части, который и становится типом результата;
- в процессе преобразования int в char лишние старшие 8 бит просто отбрасываются. Кроме того, существует возможность точно указывать требуемый тип данных, к которому необходимо привести некоторую величину (в скобках перед этой величиной). Скобки и имя типа вместе образуют операцию, называемую *приведением типов*.

Например: `z=(int)x+(int)y;`

6.4. ОПЕРАТОРЫ. УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ ЯЗЫКА

Операторы служат основными строительными блоками программы. В языке Си указанием на наличие оператора служит символ «точка с запятой», стоящий в конце него.

Операторы состоят из выражений. Выражение представляет собой объединение операций и операндов. Операндом называется то, над чем выполняется операция.

Оператор присваивания

Общий вид оператора присваивания `<Имя> = <Выражение>;`

Пример: `int x, y, a;`

`x=5;`

`y=x*2+7;`

`a=y/4;`

Переменные получают значения: `x=5, y=17, a=4.`

В языке Си разрешается включать присваивания в выражения, т. е. присваивание **может** рассматриваться как операция с учетом старшинства и влияния скобок.

Пример: `a=(y=(x=5)*2+7)/4.`

В результате переменная `x` получает значение 5, причем это значение участвует в дальнейших вычислениях. Затем выполняется умножение (`5*2`), сложение (`10+7`) и еще одно присваивание (`y=17`). Переменная `y` получает значение 17, после чего производится деление (`17/4`), результат которого присваивается переменной `a`.

В языке Си для компактной записи операторов присваивания имеются специальные операции

`+=` `-=` `*=` `/=` `%=` Так, следующие две записи на языке Си эквивалентны: `i = i+2` и `i+=2`.

Пример:

```
int x,y;
x=y=5;
x+=2; /* x увеличить на 2, результат x=7 */ y-=3; /* y уменьшить на 3. результат y=2 */
x*=y; /* x умножить на y, результат x=14 */ x/=++y; /* разделить x на увеличенный y; результат
y=3,x= 12/3 */
```

Операция присваивания сама по себе имеет значение (равное значению выражения, стоящего справа от знака "=") и может входить в выражения.

Оператор if/else

Общий вид оператора:

If(<выражение>) <оператор1>

else <оператор2>;

Здесь часть "else <оператор2>" является необязательной, можно применять и одиночный оператор

If(<выражение>) <оператор1>;

Вначале вычисляется значение выражения. Оператор выполняется, если значение выражения истинно. Если выражение ложно (его значение равно нулю), и если есть часть с else, то выполняется оператор2.

Программа 88 (нахождение наибольшего из двух целых чисел a и b)

```
#include <stdio.h>
main( )
{
    int a,b;
    printf("Введите первое число - "); scanf("%d", &a) ;
    printf("Введите второе число - "); scanf("%d", &b) ;
    if (a==b)
        printf("Заданные числа равны.\n");
    else
        if (a>b)
            printf("Первое число больше второго.");
    else printf("Второе число больше первого.");
}
```

При программировании требуется аккуратно различать знаки = и ==, потому что в ряде случаев компилятор не сможет обнаружить ошибки, связанной с неправильным использованием знаков этих операций, что приведет к неверным результатам.

В качестве оператора! может стоять любой оператор, в частности, снова оператор if/else. При этом может возникнуть неоднозначность, если во вложенной последовательности операторов if/else часть else опускается. Else всегда соответствует ближайшему предыдущему if, не содержащему else.

Например, в конструкции

```
if(n>0)
    if(a>b) z=a;
    else z=b;
```

else относится к внутреннему if. Если требуется отнести else к внешнему if, то необходимо использовать фигурные скобки:

```
if(n>0)
{
    if(a>b) z=a;
    } else z=b;
```

Часто приходится осуществлять выбор более чем из двух вариантов. Чтобы учесть это, кон-

струкция if/else расширяется конструкцией else/if. Распространенный способ выбора по значению из нескольких вариантов:

```
If (<выражение1>) <оператор1>
else if <выражение2>) <оператор2>
else if (<выражение3>) <оператор3>
else <оператор>;
```

Выражения просматриваются последовательно сверху вниз; как только какое-то выражение становится истинным, выполняется следующий за ним оператор, и на этом вся цепочка заканчивается. Последняя часть else, как и раньше, может быть опущена.

В языке Си имеется компактный способ записи одного из видов оператора if/else. Он называется «условным выражением» или «тернарной операцией». Такое выражение выглядит в общем виде так:

$B1 ? B2 : B3$

Сначала вычисляется значение выражения B1. Если оно отлично от нуля (истинно), то вычисляется значение выражения B2, которое и становится значением условного выражения. В противном случае вычисляется значение выражения B3, и оно становится значением условного выражения.

Условное выражение удобно использовать в тех случаях, когда имеется некоторая переменная, которой можно присвоить одно из двух возможных значений. Типичными примерами являются присваивание переменной значения большей из двух величин:

```
max = (a>b)?a:b;
и нахождение абсолютного значения числа:
x = (y<0)?-y : y;
```

Оператор-переключатель switch

В тех случаях, когда в программе необходимо произвести выбор одного из нескольких вариантов, удобно применять оператор switch. Его синтаксис:

```
switch (<выражение>)
{
    case <константа1>: <список операторов1>;
    case <константа2>: <список операторов2>;
    ...
    case <константаN>: <список операторовN>;
default: <список операторов>;
}
```

Оператор-переключатель выполняется следующим образом. Вычисляется значение выражения в скобках, приведенного после ключевого слова switch, затем программа просматривает список меток, указанных после слов case, до тех пор, пока не находит ту, которая соответствует данному значению. Далее программа переходит к выполнению оператора, расположенного в этой строке. Если подходящей метки не найдется и если существует строка с меткой default:, то будет выполняться оператор, помеченный этой меткой. В противном случае произойдет переход к оператору, расположенному за оператором switch.

Метки, имеющиеся в операторе switch, должны быть константами или константными выражениями (выражениями, операнды которого константы) целого или символьного типа. Запрещается использовать в качестве метки переменную. Значением выражения в скобках должна быть величина целого или символьного типа. Список операторов варианта может быть либо пустым, либо заканчиваться одним из операторов завершения (break, continue, goto, return). Если у вариантов нет общих частей, то рекомендуется каждый вариант завершать оператором break.

Программа 89

```
#include <stdio.h>
main ()
{
    int c;
    printf("Введите цифру от 1 до 7:");
    c=getchar() ;
    printf("\nСоответствующий день недели:");
    switch (c)
    {
        case '1': (printf("\nПонедельник!");break;}
        case '2': {printf("\nВторник!");break;}
        case '3': (printf("\nСреда!");break;}
        case '4': {printf("\nЧетверг!");break;}
        case '5': (printf("\nПятница!");break;}
        case '6': {printf("\nСуббота!");break;}
        default:printf("\nВоскресенье!") ;
    }
}
```

Если не использовать оператор завершения, то по окончании выполнения списка операторов выбранного варианта произойдет переход на следующий вариант из списка.

Оператор цикла for

Оператор

for (<оператор1>;<выражение>;<оператор2>) <оператор3>;

позволяет организовать повторяющийся вычислительный процесс и называется оператором цикла. Как правило, оператор1 и оператор2 являются операторами присваивания или обращения к функции, а выражение1 - условным выражением.

Цикл for удобно использовать в тех случаях, когда заранее известно количество повторений тела цикла, или имеется явно выраженная переменная, управляющая циклом. В этом случае выражение1 вычисляется один раз и задает инициализацию управляющей переменной.

Выражение1 является условием завершения цикла, а оператор2 задает приращение управляющей переменной.

Например, следующая конструкция

for (i=1; i<n; i++) <оператор>;

является широко распространенной и позволяет «перебрать» первые n натуральных чисел. Из первой строки цикла for можно сразу узнать всю информацию о параметрах цикла: начальное значение переменной i, ее конечное значение, а также насколько увеличивается значение переменной i при каждом выполнении тела цикла.

Любой из операторов и выражений в цикле for может быть опущен, хотя точка с запятой при этом должна оставаться. Если отсутствует оператор1 или оператор2, то он просто выпадает из вычислений. Если же отсутствует выражение1, то считается, что оно всегда истинно.

Например, цикл

```
for (i=1 ;;i++) {
    ...
}
```

является бесконечным.

Существуют разнообразные возможности применения цикла for:

1) можно применять операцию уменьшения для счета в порядке убывания вместо счета в порядке возрастания

```
for(n=10;n>0;n-)  
printf("%d \n",n);
```

2) при желании можно вести счет двойками, десятками и т.д.

```
for (n=2;n<60;n+=2)  
printf("%d\n",n);
```

3) можно вести подсчет при помощи символов, а не только чисел

```
for(ch='a';ch<='z';ch++)  
printf("Величина кода ASCII для %c равна %d.\n",ch,ch);
```

4) можно проверить выполнение некоторого произвольного условия, отличного от условия, налагаемого на число итераций

```
for(num=1;num*num*num<216;num++)
```

5) можно сделать так, чтобы значение некоторой величины возрастало не в арифметической, а в геометрической прогрессии

```
for(n=1;n<1500;n*=3)  
printf("-%d \n",n);
```

6) в качестве третьего выражения можно использовать любое правильно составленное выражение; какое бы выражение вы ни указали, его значение будет меняться при каждой итерации

```
for(x=1;y<=75;y=5*(x++)+10)  
printf("%d, %d\n",x,y);
```

7) можно даже опустить одно или более выражений (но при этом нельзя опускать символы ";"); необходимо только включить в тело цикла несколько операторов, которые в конце концов приведут к завершению его работы

```
ans=2;  
for(n=3;ans<=25;)  
ans=ans*n;  
Тело цикла for(;;)  
printfC"* \n");
```

будет выполняться бесконечное число раз, поскольку пустое условие всегда считается истинным;

8) первое выражение не обязательно должно инициализировать переменную; вместо этого, например, там мог бы стоять оператор printf() некоторого специального вида; необходимо помнить, что первое выражение вычисляется только один раз перед тем, как остальные части цикла начнут выполняться

```
for(printf("Запоминайте введенные числа!\n");num = 6;)  
scanf("%d",&num);  
printf("ЭТО как раз то, что я хочу!\n");
```

в этом фрагменте первое сообщение оказывается выведенным на печать один раз, а затем осуществляется прием вводимых чисел до тех пор, пока не поступит число 6;

9) параметры, входящие в выражения, находящиеся в спецификации цикла, можно изменить при выполнении операций в теле цикла; предположим, например, что есть цикл со спецификацией следующего вида:

```
for(n=1;n< 1000;n+=delta)
```

и если после нескольких итераций программа решает, что величина параметра delta слишком мала или велика, оператор if внутри цикла может изменить значение этого параметра (в диалоговой программе пользователь может изменить этот параметр в процессе выполнения цикла).

В качестве оператора в теле цикла также может быть цикл. На количество вложений циклов друг в друга ограничений нет.

В спецификацию цикла for может быть включено несколько инициализирующих или корректирующих выражений. Например, два вложенных цикла можно записать двумя различными способами:

1.

```
for(i=1;i<10;i++)  
    for(j=1;j<10;j++)  
        <оператор>
```
2.

```
for(i=1,j=1;i<10,j<10;i++,j++)  
    <оператор>
```

В нижеследующей программе переменные x, y, z изменяются параллельно.

Программа 90

```
# include<stdio.h>  
main()  
{  
    int x,y,z,v,u,zero();  
    for (x=1,y=1,z=1;x<10;x++,y++,z++)  
        printf("x=%d y=%d z=%d\n", x, y, z);  
}
```

Результат работы программы:

```
x=1 y=1 z=1  
x=2 y=2 z=2  
x=3 y=3 z=3  
x=4 y=4 z=4  
x=5 y=5 z=5  
x=6 y=6 z=6  
x=7 y=7 z=7  
x=8 y=8 z=8  
x=9 y=9 z=9
```

Оператор цикла while

В общем виде цикл while записывается так:

```
while (<выражение>) <оператор>;
```

Цикл while является «условным» циклом, использующим предусловие (т.е. условие на входе). Он называется условным, потому что выполнение оператора зависит от истинности условия, описываемого с помощью выражения. .

Если выражение «истинно» (или в общем случае не равно нулю), то оператор, входящий в цикл while, выполняется один раз, а затем выражение проверяется снова. Эта последовательность действий, состоящая из проверки и выполнения оператора, периодически повторяется до тех пор, пока выражение не станет ложным (или в общем случае равным нулю). После этого управление передается оператору, следующему за оператором цикла while.

При построении цикла while необходимо включить в него какие-то конструкции, изменяющие величину проверяемого выражения так, чтобы в конце концов оно стало ложным. В противном случае выполнение цикла никогда не завершится.

Пример 1. Алгоритм Евклида.

Программа 91

```
#include<stdio.h>
main()
{
    int x,y;
    scanf("\n%d",&x); scanf("\n%d",&y) ;
    while (x!=y)
    {
        if (x>y) x=x-y; else y=y-x;
    }
    printf("\n%d",x);
}
```

Пример 2. Проверить, содержит ли квадрат натурального числа n цифру 3.

Программа 92

```
#include<stdio.h>
main()
{
    int i,n,a;
    scanf("%d",&n); n=n*n; i=10000;
    while(i>=1)
    {
        a=n/i; /* если результат целочисленного деления n/i
                меньше 1, то a=0 */
        n=n-a*i;
        if(a==3) goto abc; else i=i/10;
    }
    printf("цифры 3 в числе n^2 нет");
    goto cd;
abc: printf("цифра 3 в числе n^2 есть");
cd: ;
}
```

Оператор цикла do/while

В языке Си имеется также конструкция цикла с постусловием (условие на выходе), где истинность условия проверяется после выполнения каждой итерации цикла. Этот подход реализуется с помощью цикла do/while.

Тело цикла do/while всегда выполняется по крайней мере один раз, поскольку проверка условия осуществляется только после его завершения.

Форма записи:

```
do <оператор>
while (<выражение>);
```

Оператор break

Оператор break дает возможность выйти из операторов цикла for, while, do/while, а также из переключателя switch без проверки условия. Оператор break приводит к немедленному выходу из самого внутреннего охватывающего его цикла или из переключателя.

Оператор break может использоваться для выхода из цикла в тех случаях, когда заданы два разных условия прекращения его работы.

Программа 93

```
# include<stdio.h>
main()
```

```

{
    int x=1,y,z;
    printf("Мы будем вычислять значение функции y=2*x+z\n");
    printf("Введите значение z:"); scant("%d",&z);
    while(x<1000)
    (
        y=2*x+z;x++; if y=100 ( printf(= 100\n"); break; }
    ) if y=100
        printf("Цикл закончен!!!") ;
    else printf ("Цикл закончен!!! Но y<>100.");
}

```

Наличие оператора break позволяет использовать «бесконечные циклы». В ряде случаев весьма удобно записать заголовок цикла без всякого условия в виде

for(;;),

а выход из цикла оформить с помощью операторов break.

Оператор продолжения continue

Оператор continue вызывает преждевременное завершение выполнения тела цикла и переход к следующему шагу цикла. Оператор continue действует только на самый внутренний цикл, частью которого он является.

Программа 94

```

#include<stdio.h>
main()
(
    int x,y,z;
    printf("Мы будем вычислять значение функции y=2*x+z\n");
    printf("в промежутках [1;6] и [13;18].");
    printf("Введите значение z= "); scant("%d",&z) ;
    for(x=1;x<18;x++)
    {
        if ((x>6) and (x<13))
            continue;
        y=2*x+z; printf(= %d, y= %d",x,y);
    }
}

```

Оператор безусловного перехода goto

Оператор перехода предназначен для безусловной передачи управления в заданную точку программы. Его выполнение заключается в передаче управления оператору, помеченному заданной меткой.

В качестве метки используется идентификатор. Метка отделяется от оператора, к которому она относится, двоеточием. Помеченный оператор должен находиться в пределах той же функции, что и оператор goto. Может быть помечен любой оператор, но никакие два разных оператора не должны иметь одинаковые метки (внутри одной и той же функции). Кроме того, один оператор может быть помечен несколькими метками.

Форма:

```

goto <метка>;...
<метка>: <оператор>

```

Составные операторы и блоки

Символы "{" и "}" используются для объединения описаний и операторов в составной оператор или блок, так что все конструкции, заключенные в фигурные скобки, оказываются синтаксически эквивалентными одному оператору. Точка с запятой никогда не ставится после первой фигурной скобки, которая завершает блок.

Составной оператор - последовательность операторов, заключенная в фигурные скобки (при необходимости его используют в качестве «внутреннего» оператора в операторах ветвления и цикла). Он называется также «блоком». Блоки не могут быть вложенными.

Пример / I -й вариант */*

Программа 95

```
#include<stdio.h>
main()
{
    int sam,index=0;
    /* В цикл включен только оператор присваивания. Печать данных */
    /* будет произведена только один раз - после завершения цикла */
    while(index++<10)
        sam=10*index+2;
    printf("sum= %d\n",sam) ;
}
```

Результат работы программы:

sam = 102

/ II -й вариант */*

Программа 96

```
#include<stdio.h>
main ()
(
    int sam,index=0;
    /* Наличие фигурных скобок гарантирует, что оба оператора */
    /* являются частью цикла while, и печать результатов будет */
    /* производиться на каждом шаге работы цикла. Весь составной*/
    /* оператор рассматривается как один оператор, являющийся */
    /* составной частью оператора while. */
    while(index++<10)
    {
        sam=10*index+2; printf("sum = %d\n",sam);
    }
}
```

Результат работы программы:

sam = 12
sam = 22
sam = 32
sam = 42
sam = 52
sam = 62
sam = 72
sam = 82
sam = 92

sam = 102

Пустой оператор состоит только из точки с запятой (;) и используется для обозначения пустого тела управляющего оператора.

6.5. СТРУКТУРА ПРОГРАММЫ НА СИ. ПОНЯТИЕ О ФУНКЦИЯХ

Программа на языке Си представляет собой набор последовательно описанных функций (процедуры и подпрограммы в языке Си считаются частным случаем функций). Каждая функция - самостоятельная единица программы, предназначенная для решения определенной задачи (или подзадачи). При описании она имеет следующий вид:

```
Тип_функции Имя (<список аргументов>)  
<описания аргументов>  
{  
  <описания>  
  <операторы>  
}
```

Отметим, что список аргументов может быть пустым (однако, скобки после имени функции сохраняются). В этом случае, естественно, нет и их описаний.

Имеется одна главная функция (с именем main), с которой начинается выполнение программы. Функции могут обращаться к другим функциям посредством конструкций вызова. Вызов функции используется при вычислении значения выражения. В результате вызова функция возвращает вычисленное значение, которое и является значением вызова функции. Попутно функция может преобразовывать значения своих аргументов. Такой результат вызова функции называется побочным эффектом.

В модуле, вызывающем данную функцию, тип возвращаемого ею значения должен быть описан (даже если это неопределенное значение) вместе с описанием переменных.

Пример. Пусть необходимо вывести на экран словосочетание «Простая функция без аргументов» 15 раз, используя функции.

Программа 97

```
#include<stdio.h>  
main ()  
{  
    int i, print() ;  
    for (i=1; i<=15; i++) print();  
    {  
        print() /* вызываемая функция без аргументов */  
    }  
    printf ("Простая функция без аргументов\n");  
}
```

Аргументы передаются по значению путем копирования в соответствующие (по порядку) параметры, указанные в определении функции. Соответствие аргументов и параметров по количеству и типу не контролируется в языке Си. Такой контроль может быть выполнен с помощью дополнительных средств отладки.

Следует различать формальные аргументы, используемые при описании функций, и фактические, задаваемые при вызове функций. Формальный аргумент - переменная в вызываемой программе, а фактический аргумент - конкретное значение, присвоенное этой переменной вызывающей программой. Фактический аргумент может быть константой, переменной или даже более сложным выражением. Независимо от типа фактического аргумента он вначале вычисляется, а затем его величина (в данном случае некоторое целое число) передается функции (см. программу 97). Можно задавать список аргументов, разделенных запятыми. *Пример:* программа 98, которая

находит числа x, y, z , принадлежащие отрезку $[1;20]$ и удовлетворяющие условию $x^2 = y^2 + z^2$.

Программа 98

```
#include<stdio.h>
main()
(
    int x, y, z, zero() ;
    char p, q, ch() ;
    x=2; y=45; z=0;
    q='o';
    printf("x» "); zero(x);

    printf("x+y+(x+y)^z= "); zero(x+y+(x+y)^z) ;
    printf("q= "); ch(q);
}
int zero(u)
int u;
(
    printf("%d\n",u);
)
char ch(u)
char u;
{
    printf("%c\n",u);
}
)
```

Результат работы программы:

```
x=2
x+y+(x+y)^z= 94
q=0
```

Программа 99

```
#include<stdio.h>
main()
(
    int x,y,z;
    int zero();
    printf("Следующие величины могут быть сторонами прямоугольного тре-
угольника:\n");
    for (x=1;x<=20;x++)
    for (y=1;y<=20;y++)
    for (z=1;z<=20;z++)
    if (y*y+z*z==x*x)
    zero(x,y,z);
}
int zero(f,g,h)
int f,g,h;
(
    printf ("x= %d, y= %d, z=%d\n",f,g,h);
)
)
```

Результат работы программы:

следующие величины могут быть сторонами прямоугольного треугольника

```
x= 5, y= 3, z= 4
x= 5, y= 4, z= 3 x= 10, y= 6, z= 8
x= 10, y=8, z=6 x= 13, y=5, z= 12
x= 13, y= 12, z= 5 x= 15, y= 9, z= 12
x= 15, y= 12, z=9 x=17, y=8, z=15 x= 17, y= 15,
```

z=8 x=20, y=12, z=16
x=20, y= 16, z= 12

Завершает выполнение данной функции и передает управление вызывающей функции **one-*ramop* return**; в главной функции main он же вызывает завершение выполнения всей программы. Оператор return может содержать любое выражение:

```
return (<выражение>);
```

Если выражение не пусто, то вычисляется его значение, которое и становится значением данного вызова функции.

Достижение «конца» функции (правой закрывающей фигурной скобки) эквивалентно выполнению оператора return без возвращаемого значения (т.е. оператор return в конце функции может быть опущен).

Пример. Данная программа вычисляет факториал, если число меньше 8; если вводимое число больше или равно 8, то выводится сообщение «Очень большое число».

Программа 100

```
#include<stdio.h>
main()
{
    int n, s() ;
    printf("Введите число ");
    scant("%d", &n) ;
    if (n<8) printf(=%d", s(n)) ;
    else printf("Очень большое число");
}

int s(x) /* определение функции с параметром */
int x;
{
    int y,p=1;
    for (y=x; y>0; y- ) p*=y;
    return(p); /* Возвращает в основную программу значение p */
}
```

Результат работы программы:

1. Введите число 4
p=24
2. Введите число 9
Очень большое число

Пример: предположим, что нужно вычислить x^2 (для некоторого неотрицательного целого y) (очевидный способ реализации возведения в целочисленную степень -многократное умножение, но существует более эффективный алгоритм, приведенный ниже).

Программа 101

```
#include<stdio.h>
main()
(
    int a, b, x, y, z;
    int odd() ;
    printf("\nВведите x, y через пробел: ");
    scanf("%d %d", &x, &y); a=x; b=y; z=1;
    while (b!=0)
```

```

    if (odd(b))
    { z=z*a; b--;}
    else
    { a=a*a; b=b/2;}
    printf("\n%d", z);
}
int odd(t)
int t;
(
    return((t%2==0)? 0:1);
)

```

Результат работы программы:

Введите x, y через пробел: 15 2
225

Если функции необходимо вернуть несколько значений, можно использовать два различных приема:

- применить глобальные переменные (в этом случае кроме изученных ранее характеристик переменных (имени, типа, значения), используется еще одна – класс памяти, см.ниже);
- применить переменные типа «указатель» в качестве аргументов функции. При вызове функции информация о переменной может передаваться функции в двух видах. Если мы используем форму обращения

```
function l(x);
```

то происходит передача в функцию значения переменной x. Изменение этого значения в вызывающую функцию не возвращается. Если мы используем форму обращения

```
function2(&x);
```

то происходит передача адреса переменной x. Теперь измененное значение переменной x может использоваться в вызывающей функции. Первая форма обращения требует, чтобы определение функции включало в себя формальный аргумент того же типа, что и x:

```
function l(num)
int num;
```

Вторая форма обращения требует, чтобы определение функции включало в себя формальный аргумент, являющийся указателем на один из объектов соответствующего типа:

```
function2(x)
int *x;
```

Обычно пользуются первой формой, если входное значение необходимо функции для некоторых вычислений или действий, и второй формой, если функция должна будет изменять значения переменных в вызывающей программе. Выше вторая форма вызова уже применялась при обращении к функции scanf(). Когда мы хотим ввести некоторое значение в переменную num, мы пишем scanf("%d",&num). Данная функция читает значение, затем, используя адрес, который ей дается, помещает это значение в память.

Пример: пусть необходимо поменять местами заданные значения переменных x и y.

Программа 102

```

#include<stdio.h>
main()
{
    int x, y;
    int interchange(); /* описание функции типа int */
    x=1; y=3;
    printf("Имели... x=1 y=3\n") ;
    interchange(&x, &y); /* обращение к функции (в данном случае передаются адреса переменных) */
    printf("Получили... x=%d y=%d", x, y);
}
/* вызываемая функция */
int interchange(u, v)
int *u, *v;
(
    int p;
    p=*u; *u=*v; *v=p;
)

```

Результат работы программы:

Имели x=1 y=3
Получили x=3 y=1

В этой программе в вызове функции `interchange(&x,&y)` вместо передачи значений `x` и `y` мы передаем их адреса. Это означает, что формальные аргументы `u` и `v`, имеющиеся в спецификации `interchange(u,v)`, при обращении будут заменены адресами `u` и `v`, следовательно, они должны быть описаны как указатели.

Поскольку `x` и `y` целого типа, `u` и `v` являются указателями на переменные целого типа, и мы вводим следующее описание:

```
int *u,*v; int p;
```

Оператор описания используется с целью резервирования памяти. Мы хотим поместить значение переменной `x` в переменную `p`, поэтому пишем: `p=*u`; Помните, что значение переменной `u` - это `&x`, поэтому переменная `p` ссылается на `x`. Это означает, что операция `*u` дает значение `x`, которое как раз нам и требуется. Мы не должны писать, например, так:

```
p = u; /* неправильно */
```

поскольку при этом происходит запоминание адреса переменной `x`, а не ее значения. Аналогично, оператор `*u = *v` соответствует оператору `x = y`.

Тип функции определяется типом возвращаемого ею значения, а не типом ее аргументов. Если указание типа отсутствует, то по умолчанию считается, что функция имеет тип `int`. Если значения функции не принадлежат типу `int`, то необходимо указать ее тип в двух местах.

1. Описать тип функции в ее определении:

```

char pun(ch,n) /* функция возвращает символ */
int n;
char ch;

```

2. Описать тип функции также в вызывающей программе. Описание функции должно быть проведено наряду с описаниями переменных программы; необходимо только указать скобки (но не аргументы) для идентификации данного объекта как функции.

```

main()
{
    char rch,pun();

```

6.6. КЛАССЫ ПАМЯТИ

Помимо изученных ранее характеристик переменных (имени, типа, значения), в ряде случаев оказывается важной еще одна - класс памяти. Класс памяти характеризует время существования и место хранения объекта в программе.

Для обозначения класса памяти в языке Си используются следующие служебные слова:

auto	extern
register	static

Автоматические объекты (auto) являются локальными по отношению к блоку и хранятся внутри того блока, где они описаны. Автоматические переменные можно инициализировать произвольными выражениями, включающими константы и ранее описанные переменные и функции.

Автоматические объекты существуют только во время выполнения данного блока и теряют свои значения при выходе из него. При каждом вхождении в блок им присваиваются начальные значения, заданные в описании. Если начальные значения не заданы, то значения автоматических объектов при входе в блок не определены. До сих пор в этом параграфе рассматривались именно автоматические объекты.

Объекты, описанные внутри блока с классом памяти register, называются регистровыми переменными. Они подчиняются всем правилам, касающимся автоматических переменных. Описание register указывает компилятору, что данная переменная будет часто использоваться. Когда это возможно, переменные, описанные как register, располагаются в машинных регистрах. Это приводит к меньшим по размерам и более быстрым программам.

Объекты, описанные внутри функции с добавлением класса памяти extern или описанные вне функции без оказания класса памяти, относятся к внешним объектам. Внешние объекты хранятся вне любой функции, входящей в состав программы, и существуют в течение выполнения всей программы. Они могут быть использованы для передачи значений между различными, в том числе и отдельно компилируемыми, функциями. Сами функции также являются внешними объектами, поскольку правила языка Си не позволяют определять одни функции внутри других. Внешние переменные можно инициализировать только выражениями с константами и указателями на ранее описанные объекты. По умолчанию (если не задана инициализация) внешние объекты получают нулевые начальные значения.

Внешние объекты делятся на внешние глобальные и внешние статические.

Важно различать описание внешнего объекта и его определение. Описание указывает свойства объекта (тип, размеры и т.д.); определение же вызывает еще и отведение памяти установке начального значения, если используется инициализация.

Например, появившиеся вне определения какой-либо функции строки

```
int max;  
char save[maxline];
```

определяют внешние переменные max и save, вызывают отведение для них места в памяти и служат в качестве описания для остальной части этого файла. В то же время строки

```
extern int max;  
extern char save[];
```

описывают в остальной части данного блока переменную max как int, а save как массив типа char (размеры которого указаны в другом месте), но не создают переменных и не отводят для них места в памяти.

Во всех блоках, составляющих исходную программу, должно содержаться только одно определение внешней переменной; другие блоки могут содержать описания extern для доступа к ней.

Программа 103

```
#include<stdio.h>
int i=0;
/* Класс памяти переменной - внешний. Область действия переменной -любая про-
грамма, */
/* загружающаяся с данным файлом. Время существования i=0 - все время выпол-
нения программы. */
main()                                     /* Блок уровня 1. */
(
    auto int i=1;
    /* В блоке 1 область действия i=1 - функция main(). Время */
    /* существования i=1 - все время выполнения главной функции */
    /* main(). */
    printf("%d\n", i) ;
    /* Если две переменные имеют одно и то же имя, то по этому */
    /* имени обращаются к внутренней переменной, внешняя */
    /* непосредственно недоступна, поэтому после выполнения */
    /* блока 1 программа напечатает i=1. */
    {      /* Блок уровня 2. */
        int i=2;
        /* Класс памяти переменной i=2 - auto. Область */
        /* действия i=2 - блок 2, время существования - время */
        /* существования блока 2. блок 2, время существования -*/
        /* время существования блока 2. */
        printf("%d\n", i) ;
        /* Программа напечатает i=2. */
        {      /* Блок уровня 3. */
            i+=1; printf("%d\n", i);
            /* Печатается самая внутренняя переменная с именем i,*/
            /* которая после выполнения операции данного блока */
            /* становится равной 3. */
        }
        /* Возвращение к блоку уровня 2. */
        printf("%d\n", i) ;
        /* Опять печатается i=3. */
    }
    /* Возвращение к блоку уровня 1. */
    printf("%d", i) ;
    /* Переменная i=3 исчезает. Теперь самой внутренней переменной */
    /* с именем i будет i=1. */
)
```

Программа 104

```
#include<stdio.h>
int a;
main()
(
    extern int a;
    int P ();
    a=6; P();
)
int P()
(
    extern int a;
    printf("a=%d",a);
)
}
```

Результат работы программы:

a=6

Областью действия статического внешнего объекта является модуль, в котором описан данный объект. При этом где-либо во внешних описаниях (т.е. вне определения функций) должно быть расположено определение внешнего статического объекта в виде

```
static <Спецификация типа> <Спецификация данных>;
```

На основании определения под объект отводится память и может быть произведена инициализация. Статические переменные можно инициализировать только выражениями с константами и с указателями на ранее описанные объекты.

При первом входе в соответствующую локальную область (блок или модуль) статические переменные инициализируются один раз (по умолчанию - нулем). При последующих входах в данную область статические переменные получают те значения, которые они имели при последнем выходе из области.

По умолчанию все функции данного модуля, расположенные ниже определения статического объекта, включаются в его область действия - в них не обязательно дополнительно описывать объект для получения к нему доступа. Функции, определения которых расположены в модуле выше определения внешнего статического объекта, для получения доступа к нему должны содержать описание этого объекта с классом памяти `extern`.

Константы являются объектами статического класса памяти.

Функция может быть определена как статический внешний объект. В этом случае она будет доступной в любой точке данного модуля и не доступной за пределами модуля.

Программа 105

```
#include<stdio.h>
main()
(
    int count;
    int trystat () ;
    for (count=1; count<=3; count++)
    (
        printf ("Итерация %d:\n", count);
        trystat() ;
    }
)
trystat ()
{
    int fade=1;
    static int stay=1;
    printf("fade = %d и stay = %d\n", fade++, stay++) ;
}
```

Результат работы программы:

```
Итерация 1:
fade = 1 и stay = 1
Итерация 2:
fade = 1 и stay = 2
Итерация 3:
fade = 1 и stay = 3
```

Если мы лишь немного видоизменим в программе функцию `trystat()`

```
trystat()
{
    int fade=1;
    int stay=1;
    printf("fade = %d и stay = %d\n", fade++, stay++);
}
```

```
}
```

то получим следующий результат:

```
Итерация 1:  
fade = 1 и stay = 1  
Итерация 2:  
fade = 1 и stay = 1  
Итерация 3:  
fade = 1 и stay = 1
```

6.7. ФУНКЦИИ ВВОДА-ВЫВОДА

Средства ввода-вывода не являются составной частью языка Си. Имеется ряд библиотечных функций Си, обеспечивающих стандартную систему ввода-вывода для программ на Си. Макроопределения, описания переменных и определения этих функций содержатся в файле стандартных заголовков `stdio.h`. Поэтому, как указывалось выше, каждая пользовательская программа должна содержать в начале ссылку

```
#include <stdio.h>.
```

В примерах программ мы неоднократно использовали форматные функции ввода (**`scanf()`**) и вывода (**`printf()`**). Набор стандартных функций ввода и вывода значительно шире и включает большое число функций для работы с данными различного типа, различными устройствами, буферизованного и небуферизованного, форматного и бесформатного ввода и вывода.

Система ввода-вывода Си обеспечивает некоторый уровень абстракции между программистом и используемым устройством. Эта абстракция называется потоком, а фактическое устройство ввода-вывода называется файлом. Буферизованная файловая система преобразует каждое физическое устройство в логическое устройство, называемое потоком. Существуют потоки двух типов: текстовые и двоичные.

Текстовый поток - это последовательность символов, которая организуется в строки, завершающиеся символами новой строки. Обработка текстового потока предполагает преобразование данных из текстового (внешнего) представления в машинное (внутреннее) или наоборот. При обработке двоичного потока последовательность его байтов взаимно однозначно соответствует байтам во внешнем устройстве.

В языке программирования Си файл - это логическое понятие, которое система может относить к чему угодно (от дисковых файлов до терминалов). Поток связывается с конкретным файлом выполнением операции «открыть». Как только файл открывается, можно обмениваться информацией между ним и программой. Закрытие выводимого потока заставляет ЭВМ записывать содержимое этого потока на внешнее устройство. Этот процесс обычно называется промыванием потока. В начале выполнения программы ЭВМ открывает три предопределенных текстовых потока `stdin`, `stdout` и `stderr`, связанных со стандартными устройствами ввода-вывода (консоль - клавиатура и дисплей). Допускается переадресация ввода-вывода к другим устройствам.

Простейшими функциями консольного ввода-вывода являются функция **`getche()`**, которая читает символ с клавиатуры, и функция **`putchar()`**, которая печатает символ на экране. Функция `getche()` ждет, пока не будет нажата клавиша, а затем возвращает ее значение, автоматически выдавая на экран «эхо» нажимаемой клавиши. Функция `putchar()` записывает ее символьный аргумент на экран в текущую позицию курсора.

Ниже приводится пример простой программы, которая принимает один символ с клавиатуры и выводит его на экран.

Программа 106

```
#include<stdio.h>  
main()  
{
```



```

char ch;
ch = getchar() ;
putchar(ch);
)

```

Есть две важные версии функции `getche()`. Первая – **`getchar()`** - буферирует ввод до тех пор, пока не введен возврат каретки. Второй версией является функция **`getch()`**, которая работает точно так же, как `getchar()`, за исключением того, что `getch()` не возвращает на экран эхо введенного символа.

Функции **`gets()`** и **`puts()`** позволяют читать и писать цепочки символов (строки) с консоли. Функция `gets()` читает цепочку символов, которая вводится с клавиатуры (ввод оканчивается возвратом каретки), помещает ее с адреса, который указывает ее аргумент - указатель символа. Функция `puts()` выводит на экран ее аргумент - цепочку символов, а затем символ новой строки. Например, нижеследующая программа читает цепочку в массив `str` и тут же печатает ее.

Программа 107

```

main ()
(
    char str[80] ;
    gets (str) ;
    puts(str) ;
)

```

Функция `puts()` занимает меньше памяти и работает быстрее, чем `printf()` при выводе символьных цепочек, поэтому программисты часто используют функцию `puts()` в тех случаях, когда важно иметь высоко минимизированный код.

Таблица 3.4
Некоторые функции буферизованной системы ввода-вывода

Имя	Функция
<code>fopen()</code>	Открывает поток
<code>fclose()</code>	Закрывает поток
<code>putc()</code>	Выводит символ в поток
<code>getc()</code>	Вводит символ из потока
<code>fseek()</code>	Ищет указанный байт в потоке
<code>fprintf()</code>	Форматный вывод в поток
<code>fscanf()</code>	Форматный ввод из потока
<code>feof()</code>	Возвращает истину, если достигается метка EOF (конец файла)
<code>ferror()</code>	Возвращает истину, если встретилась ошибка
<code>rewind()</code>	Устанавливает начальную позицию файла
<code>remove()</code>	Стирает файл

Для работы с файлами в Си используются функции буферизованной системы ввода-вывода, табл. 3.4. Обращение к ним использует указатель файла, который определяет различные характеристики файла, включая его имя, статус и текущую позицию; используется связанным с этим файлом потоком для привязки каждой функции буферизованного ввода-вывода к месту, над которым она выполняет свои операции. Указатель файла является переменной типа `FILE`, которая определяется в файле заголовков `stdio.h`.

Функция **`fopen()`** вызывается так:

```
fopen(<имя_файла>, <режим>);
```

Имя файла должно быть цепочкой символов, которая составляет правильное имя файла для операционной системы и может включать спецификацию пути. Режим задает желаемый статус открытия, табл.3.5.

Таблица 3.5
Значения режимов в Турбо-Си

Режим	Смысл
"r"	Открыть файл для чтения
"w"	Создать файл для записи
"a"	Добавлять в файл
"r+"	Открыть файл для чтения/записи
"w+"	Создать файл для чтения/записи
"a+"	Открыть или создать файл для чтения/записи

Например, для того чтобы открыть файл с именем test для записи, можно написать

```
fp = fopen("test", "w");
```

где fp -переменная типа FILE*. Переменная fp является указателем файла.

Следующий оператор обнаруживает любую ошибку при открытии файла, такую, как, например, попытку открыть защищенный от записи диск или заполненный диск, прежде чем состоится попытка записи на него:

```
if((fp = fopen("tesf", "w"))==NULL)
{
    puts("Нельзя открыть файл!\n");
    exit(1);
}
```

NULL - это макро, которое определяется в файле заголовка stdio.h.

Функция **putc()** в виде

```
puts(<символ>, fp);
```

используется для записи символа в поток, который предварительно открыт для записи с помощью функции fopen(); fp - возвращаемый функцией fopen() указатель файла.

Функция **getc()** в виде

```
getc(fp)
```

используется для чтения символов, которые она возвращает из потока, открытого в режиме чтения функцией fopen(). fp является указателем файла типа FILE, который возвращается функцией fopen(). В тех случаях, когда достигается конец файла, функция getc() возвращает маркер его конца EOF. Например, для чтения текстового файла до маркера конца файла можно использовать следующие операторы:

```
ch = setc(fp);
while(ch!=EOF)
{
    ch = getc(fp);
}
```

Функция **fEOF()** использует аргумент указателя-файла и возвращает 1, если достигнут конец

файла, или 0, если не достигнут. Например, приведенная ниже программа читает двоичный файл до тех пор, пока ЭВМ не встречает маркер конца файла:

```
while(!feof(fp)) ch = getc(fp);
```

Функцию **fdose()** используют для закрытия потока, который был открыт с помощью функции **fopen()**. Все потоки необходимо закрыть перед завершением программы. Аргументом функции является указатель файла, который закрывается.

Функции **fopen()**, **getc()**, **putc()** и **fdose()** составляют минимальный набор функций ввода-вывода. Простым примером использования функций **putc()**, **fopen()** и **fdose()** является программа, которая приведена ниже. Эта программа просто читает символы с клавиатуры и записывает их в дисковый файл до тех пор, пока не введен знак \$. Имя выходного файла задается из командной строки. Например, если вы назовете программу **ktod** («клавиша - на диск»), то набор на клавиатуре **ktod test** будет позволять вам вводить строки текста в файл с именем **test**.

Программа 108

```
#include <stdio.h>
main(argc, argv)          /*ktod - клавиша на диск */
int argc;
char *argv[];
(
    FILE *fp;
    char ch;
    if(argc!=2)
    {
        printf("Вы забыли ввести имя файла\n");
        exit(1);
    }
    if((fp=fopen(argv[1], "w"))== NULL)
    (
        printf("Не может открыть файл\n");
        exit(1);
    )
    do
    (
        ch = getchar();
        putc(ch, fp);
    )
    while (ch!='$');
    fclose (fp) ;
}
```

Еще одним примером является программа **dtos**, которая будет читать любой ASCII файл и выводить его содержимое на экран.

Программа 109

```
#include <stdio.h>
main (argc, argv)          /*dtos-wicK на экран*/
int argc;
char *argv[] ;
(
    FILE *fp;
    char ch;
    if(argc!=2) {
        printf("Вы забыли ввести имя файла\n");
        exit(1);
    }
    if((fp=fopen(argv[1], "r"))==NOLL) (
```

```

        printf("Не может открыть файл\n");
        exit(1);
    }
    ch=getc(fp);          /* читать один символ */
    while(ch!=EOF)
    {
        putchar(ch);      /* печать на экран */
        ch=getc(fp);
    }
    fclose(fp) ;
}

```

Под управлением буферизованной системы ввода-вывода можно выполнять операции чтения и записи с произвольным доступом с помощью функции **fseek()**, которая устанавливает файловую позицию.

Например, для чтения 234-го байта в файле с именем test можно использовать следующую функцию:

```

func1()
{
    FILE *fp;
    if((fp=fopen("test" ,"r"))==NULL)
    {
        printf("Не могу открыть файл\n");
        exit(1);
    }
    fseek(fp,234L,0);
    return getc(fp); /*читать один символ в 234-й позиции*/
}

```

В дополнение к рассмотренным до сих пор основным функциям ввода-вывода буферизованная система ввода-вывода включает функции **fprintf()** и **fscanf()**. Эти функции ведут себя точно так же, как функции printf() и scanf(), за исключением того, что они работают с дисковыми файлами. Обращения к функциям fprintf() и fscanf() имеют следующий вид:

```

fprintf(lp,<формат>,<список аргументов>);
fscanf(lp,<формат>,<список аргументов>);

```

где fp является файловым указателем, который возвращается вызовом функции fopen().

6.8. ДИРЕКТИВЫ ПРЕПРОЦЕССОРА

Препроцессор - это программа, которая производит некоторые, иногда весьма значительные, манипуляции с первоначальным текстом программы перед тем, как он подвергается трансляции.

Оператор препроцессора - это одна строка исходного текста, начинающаяся с символа #, за которым следуют название оператора и операнды.

Операторы препроцессора могут появляться в любом месте программы и их действие распространяется на весь исходный файл.

Весьма часто используют следующие операторы препроцессора:

```

#include
#define

```

Более специфичными являются директивы #pragma, #if, #error и др.

Важная возможность препроцессора - включение в исходный текст содержимого других

файлов. Эта возможность, в основном, используется для того, чтобы снабжать программы какими-то общими для всех данными, определениями.

Например, чрезвычайно часто в начале программы на языке Си встречается препроцессорная конструкция

```
#include <stdio.h>
```

Когда исходный текст программы обрабатывается препроцессором, на место этой инструкции ставится содержимое расположенного в некоем стандартном месте файла `stdio.h`, содержащего макроопределения и объявления данных, необходимых для работы функций из стандартной библиотеки ввода-вывода. Для использования различных математических функций необходимо подключать файл описаний `math.h`. Функции, оперирующие со строками, описаны в файле `string.h`, функции общего назначения - в `stdlib.h`, функции даты и времени - в `time.h`, диагностика - в `assert.h` и т.д.

Директива `#define` позволяет дать в программе макроопределения (или задать макросы). Оператор макроопределения имеет вид

```
#define <макроимя> <строка лексем> или  
#define <макроимя>(<список параметров>) <строка лексем>
```

Макроимя - это идентификатор. Строка лексем - это последовательность лексем от Макроимени до конца строки. Точка с запятой в конце макроопределения не ставится.

Препроцессорная обработка макроопределения сводится к тому, что любое появление Макроимени (макровызов) в качестве отдельной лексемы в тексте программы, расположенном после макроопределения, ведет к замене этого Макроимени на указанную Строку лексем.

Например, прочитав определения

```
#define PI 3.14159  
#define E 2.71828
```

препроцессор заменит в программе все имена `PI` и `E` на соответствующие числовые константы.

Препроцессор языка Си позволяет переопределять не только константы, но и целые программные конструкции. Например, можно написать определение

```
#define forever for(;;)
```

и затем всюду писать бесконечные циклы в виде `forever`

Определение макроимени с параметрами имеет некоторую специфику. Список параметров макроимени - это список идентификаторов, разделенных запятыми. Следующая после списка параметров строка лексем также может содержать эти параметры, которые при макровыводе будут заменены на соответствующие аргументы.

Макровывод должен быть отдельной лексемой и состоять из макроимени и заключенного в круглые скобки списка аргументов. При обработке макровывода препроцессор заменяет каждый параметр в строке лексем на соответствующий аргумент макровывода.

В следующих программах иллюстрируются некоторые применения операторов препроцессора.

Пример 1: найти большее из двух чисел.

Программа 110

```
#include<stdio.h>  
#define MAX(X,Y) ((X)>(Y) ? (X) : (Y))  
main()  
{  
    int x,y;  
    scanf ("%d %d", &x, &y); printf ("%d", MAX(x, y) );
```

)

Результат работы программы:

3 5
5

Пример 2.

Программа 111

```
#include<stdio.h>
#define S (x) x*x
#define P(x) printf("x равен %d.\n",x)
main()
{
    int x=4;
    int z ;
    z = S(x); P(z); z = 3(2);
    P(z);
    P(S(x));
    P(S(x+2));
    P(100/S(2));
    P(S(++x)) ;
}
```

Результат работы программы:

x равен 16.
x равен 4.
x равен 16.
x равен 14.
x равен 100.
x равен 30.

Оператор препроцессора `#pragma` позволяет записывать самые различные указания компилятору (зависящие от конкретного компилятора). Например, следующие два предложения препроцессора

```
#pragma recursive
#pragma nonrec
```

устанавливают режим всех функций программы по умолчанию рекурсивным или нерекурсивным. Указание препроцессора

```
#pragma optimize time
```

воспринимается компилятором таким образом, что он старается сгенерировать объектный код, отличающийся более высокой скоростью выполнения, чем в случае, когда он должен быть более компактным.

6.9. СИ И ПАСКАЛЬ

При знакомстве с языком Си, особенно после изучения Паскаля и Бейсика, погружение в детали его изобразительных средств может затушевать важную мысль: хотя на Си можно написать практически любую прикладную программу, он изначально для этого не предназначен. Си является результатом эволюционного развития языков создания системных программных средств. Если в прикладном программировании эволюция шла от Фортрана к Алголу, Коболу, Паскалю и т.д., то в системном - от Ассемблеров, привязанных к архитектуре ЭВМ, к Си, для которого созданы транс-

ляторы, делающие его хоть и независимым от архитектуры, но не меняющим основного предназначения.

С помощью Си можно сделать то, что на Паскале сделать невозможно (или почти невозможно) - например, написать фрагмент операционной системы (или новую операционную систему), утилиты и т.п. Так, ряд трансляторов с Паскаля написаны на Си; обратное невозможно представить. В то же время, не раз отмечалось, что прикладные программы, написанные на Паскале, отличаются большей надежностью, чем написанные на Си; их легче читать, передавать от одного программиста другому для совершенствования и сопровождения. Это связано с тем, что Паскаль содержит существенно больше ограничений и является языком более высокого уровня с сильной типизацией данных. Для языка же, который предназначен для разработки системного программного обеспечения, чем меньше ограничений, тем лучше; так, в Си возможны неявные преобразования всех базовых типов данных и указателей друг в друга, что крайне желательно при создании системных средств, но при невнимательности программиста приводит к ошибкам, не улавливаемым транслятором с Си (Паскаль же подобные недопустимые операции пресекает немедленно).

Разумеется, сказанное выше не следует абсолютизировать. Программисты, привыкшие к Си, успешно пишут на нем программы различных классов. Это касается не только Си - вспомните об экспертных системах, написанных на Бейсике. В то же время, при массовом программировании придерживаться «разделения труда» между языками представляется более естественным.

Контрольные вопросы и задания

1. Охарактеризуйте назначение и особенности языка Си.
2. Какие символы образуют алфавит языка Си?
3. Что называется лексемами, идентификаторами, литералами? Приведите примеры.
4. Какие типы данных используются в Си? Приведите примеры описания переменных.
5. Охарактеризуйте арифметические, логические и битовые операции Си.
6. Какие разновидности оператора присваивания имеются в Си?
7. Как на языке Си можно описать ветвление?
8. Охарактеризуйте возможности цикла for. Приведите примеры.
9. Какие логические циклы имеются в Си? Приведите примеры их использования.
10. Какие операторы управления имеются в Си?
11. Какова структура программы на Си? Что такое функция?
12. Приведите примеры использования функций (с аргументами и без, возвращающих и не возвращающих значения).
13. Для чего в качестве аргументов функций используются указатели? Приведите примеры.
14. Для чего в Си существуют классы памяти?
15. Что такое потоки и файлы в Си?
16. Охарактеризуйте стандартные функции ввода и вывода в Си.
17. Что такое препроцессор Си? Приведите примеры директив препроцессора.

§ 7. ОСНОВЫ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ ПРОЛОГ

7.1. ОБЩИЕ СВЕДЕНИЯ

Язык Пролог является представителем семейства языков логического программирования и в сравнении с традиционными языками программирования, предназначенными для записи алгоритмов, такими как Бейсик, Фортран, Паскаль, Си, обладает существенными особенностями:

- программа на Прологе не является алгоритмом, а представляет собой запись условия задачи на языке формальной логики (т.е. это дескриптивный, описательный язык программирования);
- язык Пролог предназначен не для решения вычислительных или графических задач, а для решения логических задач, для моделирования процесса логического умозаключения человека; вычисления же и графические построения выполняются в Прологе как побочный продукт логического вывода;
- Пролог требует особого стиля мышления программиста, что затрудняет изучение его те-

ми, кто уже привык к процедурному программированию, поэтому, так называемые, практические программисты не стремятся переходить на этот язык, что мешает росту популярности Пролога; однако во многих странах (Японии, Англии, Франции, Германии, Израиле и т.д.) расширяется практика применения Пролога в образовании как первого изучаемого языка программирования; переход к процедурным языкам типа Паскаля в этом случае трудностей не вызывает.

Все это позволяет отнести Пролог в существующем делении языков программирования на языки низкого и высокого уровня к языкам сверхвысокого уровня. В японском проекте создания в 90-х годах XX века компьютеров 5-го поколения (обладающих искусственным интеллектом) Пролог положен в основу аппаратной организации и разработки программного обеспечения. Нынешний Пролог, безусловно, не является окончательным вариантом языка программирования ЭВМ 5-го поколения и в ближайшие годы получит существенное развитие. По-видимому, он сыграет роль Бейсика дескриптивного программирования: его значение и возможности в популяризации и распространении идей логического программирования чрезвычайно велики.

Изучению языка Пролог очень способствует предшествующее изучение математической логики, понятийной системой которой он пользуется.

Программирование на Прологе включает в себя следующие этапы:

- 1) объявление фактов об объектах и отношениях между ними;
- 2) определение правил взаимосвязи объектов и отношений между ними;
- 3) формулировка вопроса об объектах и отношениях между ними.

Имена - это последовательности букв и цифр, начинающиеся с буквы (строчной !). Системы программирования на Прологе для компьютеров допускают использование лишь латинских строчных и прописных букв: *a .. z, A .. Z*. Использование русских строчных и прописных букв: *а .. я, А .. Я* не допускается. При практической работе с интерпретатором рекомендуется, чтобы смысл имен оставался понятным, использовать в качестве имен запись русских слов латинскими буквами. В данном параграфе мы будем записывать все имена русскими буквами, чтобы сделать смысл программ наиболее понятным. При запуске этих программ в «англо-язычных» системах программирования нужно заменять русские буквы в именах на латинские.

Типы данных включают переменные, атомарные значения и структуры (рис. 3.15).

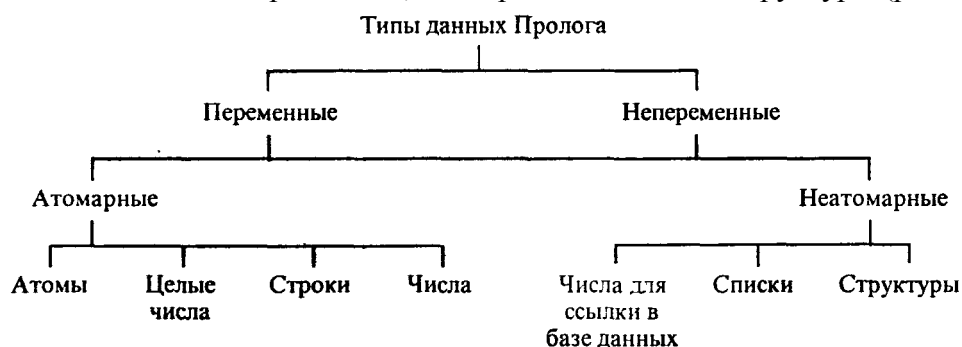


Рис.3.1 5. Классификация типов данных Пролога

Примеры *специальных атомов*:

- : - (обозначающая импликацию),
- ? (вопрос, обозначающий отрицание),
- ! (предикат отсечения, рассматривается далее).

Переменные обозначаются последовательностью буквой и цифр, начинающейся с заглавной буквы. Особый вид переменной - *анонимная переменная* `_`, используемая в качестве аргумента предиката, когда конкретное значение переменной несущественно.

Структура - это конструкция, состоящая из имени структуры и заключенного в скобки списка ее аргументов, разделенных запятыми. Элементами структур могут быть числа, атомы, переменные, другие структуры и списки. Примеры структур: `str(A,B,C)`, `носит(юрий,пиджак)`.

Списки представляют собой объединение элементов произвольных видов, разделенных запятыми и заключенных в квадратные скобки. Списки отличаются от структур тем, что количество элементов может меняться при выполнении программы. Примеры списков: `[1,3,5,7]`, `[красный,желтый,зеленый]`.

Основная операция, выполняемая в языке Пролог, - это операция сопоставления (называе-

мая также унификацией или согласованием). Операция сопоставления может быть успешной, а может закончиться неудачно. Определяется операция сопоставления так:

- константа сопоставляется только с равной ей константой;
- идентичные структуры сопоставляются друг с другом;
- переменная сопоставляется с константой или с ранее связанной переменной (и становится связанной с соответствующим значением);
- две свободные переменные могут сопоставляться (и связываться) друг с другом. С момента связывания они трактуются как одна переменная: если одна из них принимает какое-либо значение, то вторая немедленно принимает то же значение.

Примеры: 5 сопоставляется с 5, «имеет» сопоставляется с «имеет», «сергей» не сопоставляется с «юрий», «имеет(сергей,машина)» не сопоставляется с «имеет(сергей, телевизор)», «имеет(сергей,машина)» сопоставляется с «имеет(X,машина)», в этом случае переменная X получает в качестве значения атом «сергей».

Факты - это предикаты с аргументами-константами, обозначающие отношения между объектами или свойства объектов, именованные этими константами. Факты в программе считаются всегда и безусловно истинными и таким образом служат основой доказательства, происходящего при выполнении программы.

Пример 1. Факты, описывающие телефонные номера:

телефон(иванов,т561532).
телефон(петров,т642645).
телефон(сидоров,т139833).

Это означает: телефон Иванова - 56-15-32 и т.п. Заметим, что перед цифрами номера идет буква "т". Она делает номер телефона литерной константой, так как числа 561532, 642645, 139833 слишком велики, чтобы быть числовыми константами.

Пример 2, Факты, описывающие студентов:

нравится(сергей,рэп).
нравится(юрий,джаз).
носит(сергей,блейзер).
носит(юрий,пиджак).

Это означает: «Сергею нравится рэп», «Юрию нравится джаз» и т.п.

Правила - это хорновские фразы с заголовком и одной или несколькими подцелями-предикатами. Правила имеют форму

<голова правила> : - <список подцелей>

где знак : - читается «если», а список подцелей состоит из отдельных подцелей, разделенных знаком «запятая» (читаемым как «и»). Правила позволяют определить новые отношения между объектами на основе уже объявленных с помощью фактов. В качестве аргументов в предикатах правила могут использоваться не только константы, но и переменные. На переменные в правилах действуют кванторы общности, поэтому правила очень концентрированно и лаконично выражают конструкции логического вывода. Так, к фактам примера 2 можно добавить следующее утверждение:

крутойпарень(X):- нравится(X,рэп),носит(X,блейзер).

Это означает «любой X - крутой парень, если X нравится рэп и X носит блейзер». Еще примеры правил:

ест(X,Y): - пища(Y), любит(X,Y). («Каждый X ест любой Y, если Y - пища, и X любит Y»)
владелец(A,B) : - купил(A,B). («Любой A есть владелец каждого B, если A купил B»)

В Прологе все предложения программы - факты, правила, вопрос - заканчиваются точкой.

Отметим, что в Прологе вместо оператора присваивания имеется более общий и мощный механизм задания значений переменных. Переменные в Прологе получают свои значения в результате сопоставления с константами в фактах и правилах. До тех пор, пока переменная не получила какого-либо значения, она называется «свободной». Когда переменная примет значение, она становится «связанной». Однако, она остается связанной только в течение времени, необходимого для получения одного ответа на запрос, после этого Пролог «развязывает» ее, возвращается и ищет альтернативные решения. Это очень важный момент: нельзя хранить информацию, задавая значения переменных. Переменные служат частью процесса сопоставления, а не «хранилищем» информации. Область действия переменной - ровно одно предложение (правило или запрос программы).

Вопрос - отправная точка логического вывода, происходящего при выполнении программы. На любой вопрос компьютер будет пытаться дать ответ «Да» или «Нет» в зависимости от того, согласуется или нет утверждение, стоящее в вопросе, с фактами и правилами базы знаний. Вопрос, не содержащий переменных, является общим: «имеет ли место факт... ?». Так, например, к базе знаний примера 1 можно поставить вопрос

?-телефон(иванов,т123456).

и ответ будет «Нет», так как константа т123456 не согласуется ни с одним фактом.

Если к базе знаний (пример 3)

нравится(сергей,рэп).
нравится(юрий,джаз).
носит(сергей,блейзер).
носит(юрий,пиджак).
крутойпарень(X) : - нравится(X,рэп),носит(X,блейзер).

задать вопрос

?-крутойпарень(юрий).

то будет получен ответ «Нет». В самом деле, в результате резолюции утверждение в этом вопросе согласно правилу заменится конъюнкцией утверждений

нравится(юрий,рэп), носит(юрий,блейзер).

(переменная X в правиле получила значение «юрий»). Эти утверждения не согласуются с остальными фактами базы знаний. Для вопроса

? - крутойпарень(сергей).

будет получен ответ «Да», так как в этом случае противоречий при согласовании вопроса и базы знаний не возникает.

Вопрос, в котором имеются переменные, является частным: «для каких значений переменных факт ... имеет место ?». В процессе сопоставлений при выполнении программы переменные получают значения тех констант (конкретизируются), для которых сопоставление запроса, в целом, успешно, и будут выведены на экран. Так, в ответ на вопрос

? - телефон(иванов,X).

к базе знаний примера 1 на экране появится сообщение X=т561532 и будет дан ответ «Да».

Если к базе знаний примера 3 задать вопрос в форме

?- крутойпарень(A).

то свободная переменная A в вопросе сопоставляется со свободной переменной X в правиле и совмещается с ней, т.е. становится одним и тем же. В результате резолюции согласно правилу произойдет замена

крутойпарень(A)

на

нравится(A,рэп), носит(A,блейзер),

а затем предикат «нравится(A,рэп)» успешно согласуется с фактом «нравится(сергей,рэп)>>, и при этом переменная A конкретизируется значением «Сергей»; от вопроса теперь остается «носит(сергей,блейзер)», а в базе знаний имеется соответствующий факт. Ответ: «Да» и на экране появится значение присутствовавшей в вопросе переменной A:

A=сергей.

Отметим, что машина «не понимает» используемых в программе имен: «нравится», «носит», «сергей» и т.д. Мы могли бы вместо них использовать любые другие обозначения. Для интерпретатора Пролога существенны только совпадения и различия имен, а также связи между предикатами, устанавливаемые с помощью конъюнкций и импликаций. Осмысленные имена мы будем использовать только для того, чтобы облегчить чтение и понимание программ самим себе. Однако, в Прологе существуют предопределенные имена (встроенные предикаты), которые позволяют выполнить арифметические операции, сравнения, графические построения, ввод-вывод и другие полезные операции как побочный продукт выполнения программы. Встроенные предикаты Arity-Prolog описаны в справке по системе программирования, вызываемой нажатием клавиши F1.

Аналогичный набор встроенных предикатов имеется в других версиях языка Пролог.

7.2. АЛГОРИТМ ВЫПОЛНЕНИЯ ПРОГРАММ НА ПРОЛОГЕ

Факты и правила программы на Прологе являются описанием отношений и связей между объектами некоторой предметной области, т.е. записью условия некой логической задачи, которую предстоит решить. Описанные отношения и связи рассматриваются статически. Такой подход к программе называется декларативным. Порядок следования фактов, правил и подцелей в правилах не влияет на декларативный смысл программы.

Вместе с тем, программу можно рассматривать с точки зрения последовательности сопоставлений, конкретизации переменных и резолютивных выводов, происходящих при ее выполнении. Такой подход называется процедурным. Процедурный смысл программы обязательно должен учитываться при программировании на Прологе. Так, факт можно рассматривать как полностью определенную процедуру, для выполнения которой больше ничего не нужно. Правило

A:-B1,B2,...,Bn.

можно рассматривать как определение процедуры A, утверждающее, что для ее выполнения надо определить B1, B2, ... , Bn. Процедуры B1, B2, ... , Bn должны выполняться в определенном порядке - слева направо. Если выполнение очередной процедуры завершается успешно, то происходит переход к следующей процедуре. Если же по какой-либо причине очередная процедура выполняется неуспешно, то происходит переход к следующему варианту описания этой процедуры, и порядок поиска такого варианта в Прологе задан - сверху вниз. Поиск подходящих для согласования фактов и правил в базе знаний происходит последовательно сверху-вниз, и если подходящих фактов не найдено - ответ отрицательный. Эта стратегия согласования называется «сверху-вниз» и «замкнутый мир».

Рассмотрим процесс выполнения программы более подробно на примере.

Программа 112

a : - b, c, d.
b : - e, f.
c. d. e. f.
? - a.

Выполнение программы начинается с применения метода резолюций к целевому и одному из предложений программы для получения их резольвенты. Подходящее предложение программы подбирается перебором сверху-вниз так, чтобы сопоставление его заголовка с целевым предложением было успешным. В результате резолюции получается новое целевое предложение и метод резолюции применяется к нему и к другому предложению программы. Процесс продолжается до тех пор, пока не будут согласованы с фактами все возникшие при резолюции подцели, табл. 3.6.

Таблица 3.6
К процессу выполнения программы на Прологе

Номер шага резолюции	Целевое предложение	Исходное пред- ложение	Резольвента
1	?-a.	a:-b,c,d.	?-b,c,d.
2	?-b,c,d.	b:-c,f.	?-e,f,c,d.
3		?-e,f,c,d	e. ?-f,c,d.
4		?-f,c,d.	f. ?-c.d.
5		?-c,d.	c. ?-d.
6		?-d.	d. Пустая

При выполнении логического вывода, если необходимо, происходит конкретизация переменных. Рассмотрим пример.

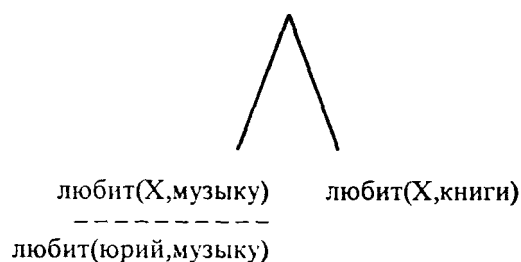
Программа 113

любит (юрий, музыку) .
любит (сергей, спорт) .
любит (А, книги) : -читатель (А) , любопытный (А) .
любит (сергей, книги) .
любит (сергей, кино) .
читатель (юрий) .
любопытный (юрий) .
?- любит (X, музыку) , любит (X, книги) .

Двойной запрос в этой программе может быть представлен целевым деревом:



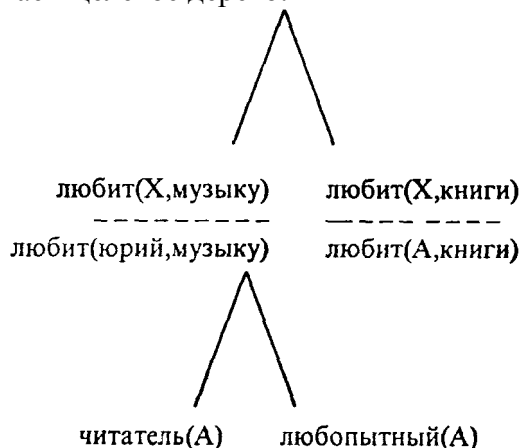
Вначале, просматривая программу сверху вниз. Пролог находит первое предложение, соответствующее первой подцели запроса:



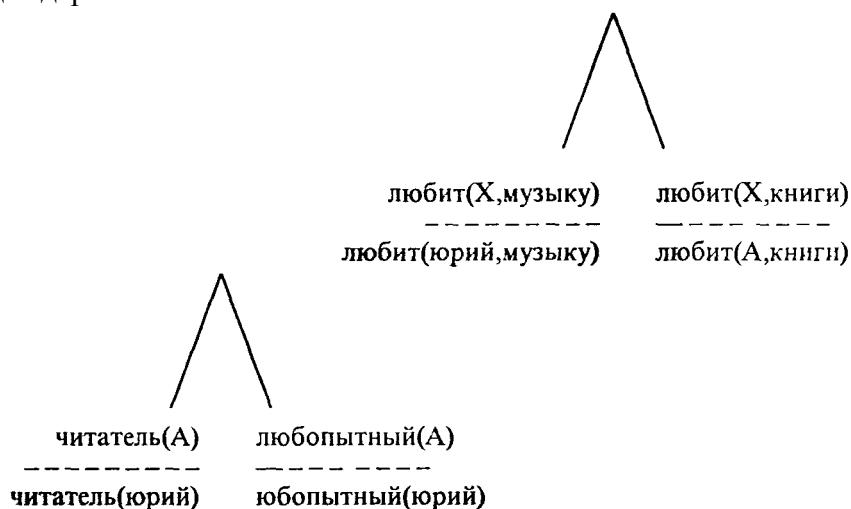
Переменная X конкретизируется значением «юрий». Начинается согласование 2-й подцели запроса с условием X=юрий. 1-е и 2-е предложения программы не соответствуют подцели. В 3-ем предложении:

любит(A,книги):-читатель(A), любопытный(A).

аргумент A заголовка есть переменная, поэтому она может соответствовать X, т.е. получает значение A=юрин; вторые аргументы совпадают. Теперь тело правила образует новое множество целей для согласования. Получаем целевое дерево:



Затем Пролог будет искать факты, соответствующие новым подцелям. Последнее результирующее дерево:



Рассмотрим еще один пример.

Программа 114

любит (оля, чтение) .
 любит (света, бадминтон) .
 любит (для, бадминтон) .
 любит (лена, плавание) .
 любит (лена, чтение) .
 ?- любит (X, чтение) , любит (X, плавание) .

Запрос означает: есть ли люди, которым нравится и чтение, и плавание? Сначала Пролог ищет факт, сопоставимый с первой частью вопроса: любит(X, чтение). Подходит первый же факт программы

любит(оля,чтение).

и переменная X связывается значением «оля». В то же время Пролог фиксирует в списке фактов указатель, показывающий состояние процедуры поиска. Далее Пролог пытается согласовать вторую часть запроса при условии X = оля, т.е. ищет с самого начала программы факт «любит(оля, плавание)». Такого факта в программе нет, и поиск заканчивается неуспешно. Тогда Про-

лог возвращается к первой части запроса: $\text{любит}(X, \text{чтение})$, «развязывает» переменную X и продолжает поиск подходящих фактов, начиная с ранее установленного в списке фактов указателя. Подходит факт «любит(лена, чтение)», переменная X конкретизируется значением «лена», и далее вторая часть вопроса успешно согласуется с фактом «любит(лена, плавание)». Пролог выполнил в данном примере поиск с возвратом.

Графически процесс выполнения программы представляется в виде обхода бинарного дерева - дерева вывода, типа изображенного на рис.3.16. Вершины дерева обозначают вопросы, а ребра показывают возможные пути вывода, причем для каждого ребра характерны свои правила и унифицирующая подстановка значений переменных.

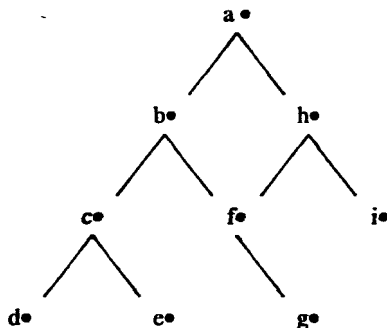


Рис.3.16. Дерево вывода программы на Прологе

Обход дерева начинается с движения от вершины (запроса) по самой левой ветви вниз до конца (abed), при этом запоминаются все точки ветвления (точки возврата). При достижении конца ветви решение будет либо найдено, либо нет. В обоих случаях Пролог продолжает дальнейший поиск решений. Выполняется возврат в последнюю точку ветвления c . При этом конкретные значения, присвоенные переменным при движении вниз на сегменте $c-d$, отменяются, и движение вниз продолжается по расположенной справа ветви $c-e$ до конца дерева вниз. Затем произойдет возврат в предыдущую точку ветвления b и движение продолжится по ветви bfg , и так до тех пор, пока все дерево вывода не будет пройдено.

7.3. РЕКУРСИЯ

Существует целый класс задач, в которых отношения между объектами можно определить, только пользуясь самими определяемыми соотношениями. Получающиеся при этом правила называются рекурсивными.

Пример: рекурсивное определение натурального числа:

- 1) 1- натуральное число;
- 2) число, на 1 большее натурального числа, также натуральное.

В системах логического программирования рекурсия служит также для описания циклов, повторений и является важнейшим методом программирования.

Рассмотрим простой пример: вычисление факториала натурального числа n ($n!$). Определение $n!$ рекурсивно:

- 1) $1! = 1$,
- 2) $n! = (n-1)! * n$.

Для описания отношения «факториал» между n и $n!$ будем использовать двухарный предикат

$\text{факт}(N, M)$. Тогда база знаний, соединенная с запросом, приобретает вид (программа 115);

Программа 115

```
факт(1, 1).
факт(N, X) :- факт(N-1, V), X is V*N.
?- факт(3, A);
```

В данной программе правило «факт» вызывает само себя - это и есть рекурсия. Запись $\text{is } Y*N$ представляет собой обращение к встроенному предикату «is» («есть») для описания арифме-

тического действия.

Процесс работы программы можно изобразить следующим образом:

?факт(3,A0).

ОТВЕТ: A=6

?факт(1,A2).

X1= 2*3 = 6

факт(1,1)

X2=1*2=2

Здесь стрелочка вниз означает сопоставление и резолюцию, а стрелочка вверх - возврат и завершение отложенного вычисления.

Правило «факт» вызывает само себя - происходит углубление рекурсии (прямой ход). При этом в памяти ЭВМ выделяется место для переменных A,AO,A1,A2 и N,NO,N1,N2, образующих стеки. При согласовании вопроса с предикатом факт(1,1) рекурсия прекращается и начинается возврат из рекурсии (обратный ход) - выполнение отложенных на прямом ходе согласований. Предикат факт(1,1) играет очень важную роль - это ограничитель рекурсии, условие ее завершения.

Отметим, что Пролог стремится найти все решения поставленной задачи, а значит, после появления ответа A=6 происходит возврат к вопросу ?факт(1,A2) и попытке согласовать его с правилом «факт». Это приводит к бесконечному процессу рекурсии с отрицательными аргументами в «факт», которая завершается при исчерпании глубины зарезервированных интерпретатором Пролога стеков. Ускорить выход из рекурсии можно, добавив к предикату «факт(1,1)» отсечение !:

факт(1,1):-!.

Однако, использование отсечения требует более подробного рассмотрения. В общем случае последовательность предложений в базе знаний не имеет значения. Однако это не так для рекурсивно-определенных отношений. Например:

родитель(X):- родитель(Y),отец(Y,Z).

родитель(коля).

отец(коля,петя).

родитель(петя).

В этом случае в первом предложении голова имеет ту же функцию, что и одна из целей - «родитель». В процессе поиска ответа в этой базе знаний будет применено правило: **предложение, стоящее первым, будет применено первым** - известное как *принцип поиска в глубину*.

Это приведет к тому, что система будет обращаться только к первому предложению базы знаний и ответ на вопрос не будет найден никогда (образуется бесконечная петля вывода). Однако небольшое изменение базы знаний - перестановка двух предложений местами - приведет к удачному поиску решения.

Программа 116

родитель (коля) .

родитель (X) :- родитель (Y) , отец (Y, X) .

отец (коля, петя) .

? - родитель (петя) .

Неограниченно-повторное обращение к предложению может быть и более замаскированным так, как это получается в программе 117.

Программа 117

выше(A,B): - ниже(B,A).

ниже(B,A): - выше(A, B).

выше(коля,петя).

?- ниже(петя,коля).

Однако если третье предложение стоит на первом месте, то повторного обращения не произойдет и ответ будет найден.

В общем виде рекурсия на Прологе выглядит так:

$P(1,...)$.

$P(n,...) \text{ - } Q_1,..., Q_n, P(n-1,...), R_1,... R_m$.

Правило P обращается само к себе, при этом происходит углубление рекурсии. Предикаты $Q_1, ..., Q_n$ выполняются на прямом ходе рекурсии, а $R_1,..., R_m$ - на обратном; n - это некоторый условный параметр, входящий в условие продолжения рекурсии, а $P(1,...)$ - факт, завершающий процесс рекурсии.

Особенно простым случаем рекурсии является простое циклическое повторение. Один из способов организации повторения связан с наличием в базе знаний процедуры вида `repeat`, `repeat`: - `repeat`.

Использование `repeat` в качестве подцели некоторого правила приводит к многократному повторению остальных подцелей этого правила.

7.4. ПРЕДИКАТ ОТСЕЧЕНИЯ И УПРАВЛЕНИЕ ЛОГИЧЕСКИМ ВЫВОДОМ В ПРОГРАММАХ

Управление процессом просмотра предложений является важным аспектом программирования на Прологе. Это осуществляется с помощью специальной встроенной функции «резать», обозначаемой символом `!`.

Данная встроенная функция может быть использована для достижения следующих трех целей:

- 1) исключения бесконечной петли при выполнении программы;
- 2) программирования взаимоисключающих утверждений;
- 3) блокирования просмотра целей.

Продemonстрируем все три случая на примерах.

Пример 1. Устранение бесконечных циклов. Обратимся к утверждениям, определяющим последовательность Фибоначчи (числовая последовательность 1, 1, 2, 3, 5, 8,..., в которой каждое число, начиная с третьего есть сумма двух предыдущих).

Программа 118

```
fib (0,_,1) .  
fib (1,1,1) .  
fib (N,G,H) : - fib ( N-1 ,F,G), H is F+G.
```

На запрос

?- fib (0_,F).

получим $F = 1$, и Пролог сделает попытку сопоставить с запросом второй факт и потерпит неудачу. Однако сопоставление головы третьего утверждения с запросом происходит успешно и осуществляется попытка доказать цель `fib(-1,FO,Fl)`, что, в свою очередь, приводит к цели `fib(-2, ..., ..)` и так далее, т.е. образуется бесконечный цикл.

Однако мы можем устранить такие ситуации, используя отсечение и тем самым указывая Прологу, что не существует других решений в случае успешного согласования граничного условия.

Программа 119

```
fib (0,_,1) : - !.  
fib (1,1,1) : - !.  
fib (N,G,H) : - fib ( N-1 ,F,G), H is F+G.
```


Учитывая данное определение fib и задавая вопрос

?- fib(0_ ,F).

получаем F=1. Других решений нет.

Пример 2. Программирование взаимоисключающих утверждений. Процедуру нахождения наибольшего из двух чисел можно записать в виде отношения

max(X, Y, M).

Здесь M=X, если $X \geq Y$, и M=Y, если $X < Y$. Соответствующие правила таковы:

max(X,Y,X):-X>=Y.

max(X, Y, Y) : - X<Y.

Эти правила являются взаимоисключающими. Возможна более экономная формулировка, использующая понятие «иначе»:

если $X \geq Y$ то M=X иначе M=Y.

На Прологе это записывается при помощи отсечения:

max(X,Y,X):-X>=Y,!

max(X, Y, Y).

Пример 3. Блокирование просмотра целей.

Программа 120

B.

D

A: - B, C. (1)

C: -D, !, E. (2)

E: -F, G, H. (3)

?A.

Говорят, что дизъюнкт (1) «порождает» дизъюнкт (2), так как в правой части (1) есть литера C и эта же литера - в левой части (2). Аналогично дизъюнкт (2) «порождает» дизъюнкт (3). Если (3) неудачен, то в (2) выполнится отсечение: дизъюнкт (2) также считается неудачным, восстанавливается «родительская среда» (1), делается попытка найти альтернативное решение для B. Если бы (2) имело вид C: -D, E. , то при неудаче в (3) была бы сделана попытка найти альтернативное решение для D.

В других случаях может быть необходимым продолжение поиска дополнительных решений, даже если целевое утверждение уже согласовано. В этих случаях можно использовать встроенный предикат fail.

Встроенный предикат fail не имеет аргументов. Он считается всегда ложным.

Пример: перебор всевозможных решений.

Программа 121

ос(cpm) .

ос(msdos) .

ос(unix) .

печать-всех:-ос(X), write(X), fail.

?-печать-всех.

7.5. ОБРАБОТКА СПИСКОВ

На практике часто встречаются задачи, связанные с перечислением объектов. В некоторых случаях при решении задач важно сохранять информацию об уже сделанных шагах решения, чтобы их не повторять. Для решения таких задач в языке Пролог предусмотрены списки.

Список можно задать перечислением элементов. Например, имена учеников класса:

[саша,петя,дима,ксюша,лена].

Элементами списка могут быть не только атомы, но и функции, и вообще любые элементы, даже списки. Заранее длина списка не задается, и в ходе выполнения программы она может меняться.

Альтернативный способ задания списка использует понятия головы и хвоста списка.

Например, в списке $[X | Y]$ X - это голова списка. Y - его хвост.

Хвост списка по определению также является списком.

Теперь список может быть определен рекурсивно:

1) пустой список $[]$ - список:

2) $[X | Y]$ - список, если Y - список.

Определение списка через его голову и хвост в сочетании с рекурсией лежит в основе большого числа программ, оперирующих списками. Эти программы состоят

1) из факта, ограничивающего рекурсию и описывающего операцию для пустого списка;

2) из рекурсивного правила, определяющего операцию над списком, состоящим из головы и хвоста (в голове правила), через операцию над хвостом (в подцели).

Пример 1: определение числа элементов в списке.

Программа 122

сколько $([], 0)$.

сколько $([A|B], N) :-$ сколько $(B, M), N \text{ is } M+1$.

?- сколько $([саша, игорь, лена], X)$.

Ответ: $X=3$.

Пример 2: принадлежность элемента списку.

Программа 123

принадлежит $(X, [X | Y])$.

принадлежит $(X, [A | Y]) :-$ принадлежит (X, Y) .

?-принадлежит $(4, (1, 3, 4, 9))$.

Ответ: да.

Данная программа имеет очень простой декларативный смысл: элемент принадлежит списку, если он является его головой или принадлежит хвосту списка.

Пример 3: соединение двух списков.

Эту задачу можно описать с помощью следующих предикатов:

а) ограничение рекурсии состоит в том, что если к пустому списку $[]$ добавить список P , то в результате получится P ;

б) рекурсия состоит в том, что можно список P добавить к концу списка $[X|Y]$, если P будет добавлен к хвосту Y и затем присоединен к голове X (при этом получается список $[X|T]$).

Программа 124

присоединить $([], P, P)$.

присоединить $([X|Y], P, [X | T]) :-$ присоединить (Y, P, T) .

? присоединить $(L, [джим.R], (джек, бил, джим, тим, джим, боб))$.

Ответ:

$L = [джек, бил]$. $K = [тим джим, боб]$.

$L = [джек, бил, джим, тим]$. $R = [боб]$.

Существует традиция использовать для обозначения предиката слияния двух списков предикативный символ `append` (по-английски -добавить).

В некоторых случаях постановки вопросов к такого рода программам приходится исполь-

зовать отсечение (!).

Программа 125

```
append([ ], L, L).  
append([A | B], C, [A | D]):- append(B, C, D).  
?-append(X, Y, [1, 2]).
```

Ответ:

X=[]

Y=[1,2]

X=[1]

Y=[2]

X=[1,2]

Y=[].

Если же заменить первое предложение на `append([ ], 1,1):- !.` и задать тот же вопрос, то получится правильный ответ:

X=[]

Y=[1,2].

Пример 4. удаление элементов из списка.

Программа 126 аналогична проверке принадлежности элемента списку, но требует уже трехарного предиката, один аргумент которого указывает удаляемый элемент, второй аргумент-исходный список и третий - список-результат.

Программа 126

```
удал (X. [X | Y], Y) : - !.
```

```
удал (X. [Z | Y], [Z | W]) : - удал (X, Y, W) .
```

Декларативный смысл: если удаляемый элемент совпадает с головой списка, то результатом программы является хвост списка, иначе удаления производятся из хвоста списка.

Данная программа удаляет первое вхождение в список элемента, связанного с переменной X. Знак отсечения "!" в конце правила предотвращает последующий поиск и вывод лишних вариантов ответов после выполнения ограничительного факта.

Для удаления всех вхождений элемента X программу надо дополнить:

```
удал (X,[ ],[]).
```

```
удал (X, [X | Y], W) :- удал (X, Y, W).
```

```
удал (X, [Z | Y], W):- удал (X, Y, W).
```

Декларативный смысл программы таков: пока список не пуст, удалить элемент, если он совпадает с головой списка, значит, отбросить голову списка, а затем удалять его из оставшегося хвоста, иначе надо сразу удалять элемент из хвоста.

Пример 5: индексация элементов списка.

Смысл программы 127 состоит в том, чтобы получить элемент под номером N или узнать номер элемента X.

Программа 127

```
получить ([X | Y], 1, X) .
```

```
получить ([W | Y], N, X) :- N is M+1, получить (Y, M, X) .
```

Пример 6: поиск максимального элемента.

Программа 128

```
max ([X], X) .
```

```
max ([X | Y], X) :- шах (Y, W), X>W, !.
```

```
max ([X | Y], W) :-max (Y, W) .
```

Декларативный смысл программы: если в списке один элемент - он и является максималь-

ным, если более одного, то это голова списка, если она больше максимального элемента хвоста, или максимальный элемент хвоста.

Пример 7: обращение списка.

Данная задача - самая сложная из рассмотренных. Для ее решения важно сообразить, что обратить список из одного элемента - означает оставить список без изменения. Обратить более длинный список - обратить его хвост, а потом сзади приставить к нему голову исходного списка.

Программа 129

обр ([X], [X]) .

обр ([X I Y], Z) :- обр (Y, W), присоединить (W, [X], Z) .

В этой программе используется процедура слияния списков, описанная выше.

Arity-Prolog располагает значительным числом встроенных предикатов для обработки списков, так что приведенные программы имеют, в основном, учебный характер.

7.6. РЕШЕНИЕ ЛОГИЧЕСКИХ ЗАДАЧ НА ПРОЛОГЕ

Целью всего предшествующего изложения была подготовка к данному разделу - решению содержательных логических задач на Прологе, т.е. задач невычислительного характера, в которых особенности Пролога и дескриптивной парадигмы программирования проявляются наиболее ярко.

Рассмотрим пример: нарисовать конверт, не отрывая карандаша от бумаги и не проводя два раза по одной и той же линии.

Введем обозначения, как показано на рис. 3.17. Ребра графа обозначены буквами а, б, в ... (литерные константы), вершины - цифрами 1, 2, 3 ... Опишем структуру графа предикатом вида «ребро (S, A, B)», что означает, что от вершины A к вершине B идет ребро S. Так как граф неориентированный, помимо предикатов вида «ребро (S, A, B)» нужны и предикаты «ребро (S, B, A)». Знания о структуре графа можно представить так, как это записано рядом с рис. 3.17.

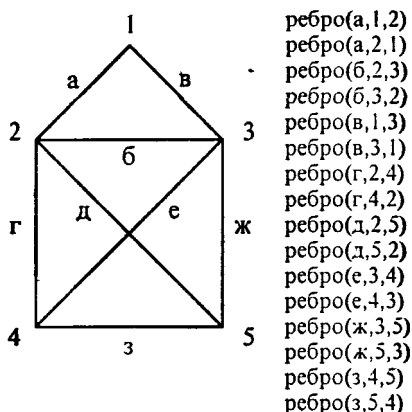


Рис. 3.17. Задача «конверт»

Решением задачи должен явиться список пройденных ребер графа, причем длина его должна быть равна 8 и в нем не должно быть повторяющихся ребер, что можно описать так:

путь(Т,П) : - длина(П,8), write_list(П),!. (1)

путь(Т,П) : - ребро(Р,Т,Н),не_принад(Р,П),путь(Н,[Р|П]). (2)

Переменная Т обозначает текущую вершину графа, а П - список пройденных ребер. Правило 1 означает, что если длина списка П пройденных вершин становится равной 8, список П выводится на печать. Это правило ограничивает рекурсивный перебор вершин и ребер, проводимый правилом 2. Правило 2 является генератором перебора, оно перебирает предикаты «ребро()» и находит такое ребро Р из текущей вершины Т в новую Н, чтобы оно не принадлежало списку П, затем это ребро добавляется в качестве головы к списку П, и поиск дальнейшего пути производится уже из новой вершины Н.

Нам потребуется программа, определяющая длину списка,

длина ([],0).

длина ($[A \mid B], N$) :- длина (B, M), N is $M+1$.

а также программа вывода элементов списка на экран

```
write_list([]).  
write_list([H | T]):-write(H),write_list(T).
```

Задание

?-путь(4,[]).

- искать путь, начиная с вершины 4 и пустого списка пройденных ребер.

Ответ: з, ж, в, а, б, д, г, е.

На вопрос ?-путь(1,[]) ответ-«НЕТ».

Аналогично решаются другие задачи, связанные с поиском пути в графе, удовлетворяющего каким-то дополнительным условиям, например задача о коммивояжере. Программа будет состоять

- 1) из базы знаний о структуре графа - вершинах и связывающих их ребрах (каждому ребру может сопоставляться набор весов);
- 2) из правил, выражающих дополнительные условия и ограничения на решения задачи и часто связанных с обработкой списков.
- 3) из рекурсивного правила - генератора перебора ребер и вершин с некоторым ограничивающим предложением, целевым условием;
- 4) из дополнительных процедур и промежуточных определений.

Интересно, что большинство задач, которые считаются логическими, сводятся к задаче поиска пути в некотором графе - графе состояний задачи. К этому типу задач можно отнести и разнообразные игры. Характерными особенностями многих задач являются следующие:

- 1) наличие неких дискретных состояний, число которых конечно, и одно из них принимается за начальное, а другое (или несколько других) за конечное (искомое);
- 2) определены правила перехода между состояниями;
- 3) для каждого состояния заданы определенные условия допустимости (оценки) этого состояния.

При анализе предметной области задачи эти состояния, правила перехода и условия допустимости должны быть выявлены, получены соответствующие обозначения и затем записаны с помощью фраз Хорна.

Рассмотрим задачу: имеются два сосуда - на 3 и на 5 литров. Как отмерить с их помощью 4 литра воды ?

В этой задаче состояния связаны с определенным количеством воды V в первом сосуде и W во втором. Начальным состоянием является $V=0, W=0$, а конечным $V=3, W=4$. Переходы между состояниями можно записать в виде правил:

```
сосуды(V1, W1):- сосуды(V2, W2).
```

Например, правило

```
сосуды(0, W) :- сосуды(V, W).
```

означает, что вся вода из первого сосуда вылита. Обратим внимание на слово «вода» в условии задачи. Для предметной области, связанной с водой, характерно то, что воду можно просто выливать, и данное правило перехода между состояниями допустимо. Если бы задача решалась для молока, то его выливать было бы нельзя, и такое правило было бы недопустимым !

Правило

сосуды(3, W) :- сосуды(V, W). означает, что первый сосуд заполнен полностью.

Не разливая, жидкость можно перелить из одного сосуда в другой только так, что один станет пустым, а другой наполнится. Это можно записать в виде правил

сосуды(3, W):- сосуды(V, W-V+3).
сосуды(V, 0):- сосуды(V-W, W).
сосуды(V, 5): - сосуды(V-W+5, W).
сосуды(0, W):- сосуды(V, W-V).

При решении данной задачи необходимо также избежать повторения одних и тех же состояний - «переливания из пустого в порожнее». Для этого в предикат «сосуды ()» следует добавить 3-й аргумент - список пройденных состояний П. Элементы в него будут добавляться парами:

сосуды(V1, W1, [V1, W1|П]):- не_принад(V1, W1, П), сосуды(V2, W2, П).
Условие, ограничивающее рекурсию, должно иметь вид:
сосуды(_, 4, П) :- write_list(П).

Контрольные вопросы и задания

1. В чем состоят принципиальные различия процедурных и декларативных языков программирования?
2. Каковы этапы программирования на Прологе?
3. Какие типы данных допускает Пролог?
4. В чем сущность операции сопоставления?
5. Как реализуются вопросы к программе на Прологе?
6. Приведите примеры рекурсий, отличные от данных в тексте.
7. Для чего служит предикат отсечения?
8. Для чего служат списки и как они задаются?
9. Опишите на Прологе:
 - а) свою родословную, определите бабушек, дедушек, прабабушек, прадедушек и т.д.;
 - б) телефонную книгу;
 - в) районы вашего города, республики, области, укажите численность их населения, местные достопримечательности;
 - г) европейские государства (население, площадь и т.д.);
 - д) таблицу дат и событий русской истории;
 - е) небольшой словарь для перевода с русского языка на иностранный язык, который вы изучаете;
 - ж) ведомость зачета вашей группы;
 - з) успеваемость вашей группы (дайте определение «отличника»);
 - и) каталог книг в библиотеке.
10. Запишите на Прологе правила, являющиеся решением следующих заданий:
 - а) даны два числа а и b, получите их сумму, разность, произведение;
 - б) дана длина ребра куба, найдите объем куба и площадь его боковой поверхности;
 - в) дан радиус основания r и высота цилиндра h, найдите его объем и площадь боковой поверхности;
 - г) даны стороны а и b параллелограмма, а также угол между ними, найдите диагонали параллелограмма и его площадь;
11. Вычислите значения выражений:
 - а) $2x+3y+4$
 - б) $(2x+8y+4)/2$
 - в) $y-x^2$
 - г) x^2+xy+y^2
 - д) $x/2+5y$
 - е) x^2+3y^2
 - ж) $5(34x-y)$

12. Напишите программы, выполняющие операции над списками:
 - а) объедините два списка, найдите MAX и удалите его;
 - б) удалите из списка элемент, найдите длину оставшегося списка;
 - в) добавьте к списку элемент, вычислите среднее арифметическое его элементов;
 - г) обратите список, найдите последний и предпоследний элементы;
 - д) исключите из списка заданный элемент во всех вхождениях, кроме первого, найдите длину оставшегося списка;
 - е) проверьте, имеются ли в списке повторяющиеся элементы, и все их удалите;
 - ж) удалите из списка все элементы, равные последнему, найдите длину оставшегося списка;
 - з) объедините два списка, найдите MIN и удалите его;
 - и) обратите список, найдите MAX и удалите его;
 - к) к списку добавьте обращенный второй список, найдите длину результата;
 - л) отсортируйте список, используя метод пузырька.
13. Напишите программу, решающую задачу о волке, козе и капусте.
14. Напишите программу, решающую задачу, аналогичную задаче 13, о трех туземцах, трех миссионерах и двухместной лодке.
15. Напишите программу, решающую задачу об обходе препятствия.
16. Напишите программу, определяющую положение «шах королю».
17. Напишите программу, определяющую, как шахматному коню попасть с поля А на поле В.
18. Напишите программу, определяющую, как разлить 10 л молока по 5 л, пользуясь бидонами на 3, 7 и 10 л.
19. Напишите программу, аналитически дифференцирующую элементарную функцию.
20. Напишите программу, играющую в «крестики и нулики» на бесконечной плоскости.
21. Напишите программу, вычисляющую интервал между двумя датами одного года, например 7 марта и 9 сентября.
22. Напишите программу, решающую задачу о четырех ферзях на поле размером 4x4 клетки.

§ 8. ВВЕДЕНИЕ В ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ЛИСП

8.1. НАЗНАЧЕНИЕ И ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКА

В программировании помимо процедурного подхода, представителями которого являются такие универсальные языки высокого уровня как Бейсик, Паскаль, Си, и логического подхода, представленного языком Пролог, существует еще одно направление - функциональное. Оно возникло в 1962 г. вместе с созданием Дж.Маккарти языка программирования Лисп (Lisp). Долгое время этот язык занимал особое место. Подавляющее большинство программ искусственного интеллекта составлено на языке Лисп. До сих пор он считается стандартным языком разработки систем искусственного интеллекта. Его популярность особенно велика в США. В нашей стране этот язык не получил широкого распространения (одна из причин - недостаток литературы о нем на русском языке), однако в настоящее время популярность этого языка быстро растет. Несмотря на то, что Лисп - один из самых старых используемых языков программирования, у него многое еще впереди.

Язык Лисп - один из первых языков обработки данных в символьной форме. Его название происходит от английских слов «list processing» - «обработка списков». В Лиспе и программа, и обрабатываемые ею данные представляются в одной и той же форме - в форме списка. Таким образом, программы могут обрабатывать и преобразовывать другие программы и даже самих себя.

Используемый в Лиспе, так называемый, функциональный подход к программированию основывается на той простой идее, что вся обработка информации и получение искомого результата могут быть представлены в виде вложенных и/или рекурсивных вызовов функций, выполняющих некоторые действия, так что значение одной функции используется как аргумент другой. Значение этой функции становится аргументом следующей и т.д. пока не будет получен конечный

результат - решение задачи.

Программы строятся из логически расчлененных определений функций. Определения состоят из управляющих структур, организующих вычисления, и из вложенных вызовов функций. Основными методами функционального программирования являются композиция и рекурсия. Все это представляет собой реализацию идей теории рекурсивных функций.

Имеется большое число систем программирования на Лиспе, реализованных для компьютеров различных типов. Как правило, это интерпретирующие системы, работающие в интерактивном (диалоговом) режиме. Соответствующие описания и команды вводятся с клавиатуры после приглашения ("_"), затем прочитывается результат.

8.2. ОСНОВНЫЕ ЭЛЕМЕНТЫ ПРОГРАММЫ НА ЛИСПЕ. СПИСКИ

Программы на языке Лисп строятся из простейших неделимых элементов, называемых атомами. Символы и числа представляют собой атомы, из них состоят все остальные структуры.

Символ - это имя, состоящее из букв, цифр и специальных знаков, которое обозначает какой-нибудь предмет или действие из реального мира, а также число, функцию (программу) и другие объекты. Наряду с символами используются и числа (значения), которые могут быть целыми (например, 543), десятичными (например, 3.789) и в представлении с мантиссой и порядком (например, 1.0243E-6).

Главной структурой в Лиспе является список.

Списком называется упорядоченная последовательность, элементами которой являются либо атомы, либо списки (подсписки). Списки заключаются в круглые списки, а их элементы разделяются пробелами. Например,

```
(ab(cd)e)
(В группе 18 студентов)
((((первый) 2) третий) 4) 5).
```

Список, в котором нет ни одного элемента, называется пустым списком и обозначается "()" или специальным символом NIL. Список - это структура данных, представляющая некоторую иерархическую связь (дерево) с помощью строго соответствующих друг другу открывающих и закрывающих скобок.

Имеется и альтернативный способ записи списков - с использованием, так называемой, точечной нотации. Точка при этом отделяет начальный элемент списка - его голову - от остальной части списка - хвоста: (голова, хвост) или

```
(a1 a2 ... aN) = (a1. (a2.... (aN.NIL)...)).
```

Здесь NIL - это предопределенная константа, означающая пустой список (и одновременно логическое значение «Ложь»).

Атомы и списки называются **S-выражениями**. Все вышесказанное можно обобщить в следующих формах Бэкуса - Наура

<S-выражение>	:: = <атом> <список>
<список>	:: = (<внутренняя часть>)
<внутренняя часть>	:: = NIL <S-выражение> [{внутренняя часть} }
<атом>	:: = цепочка алфавитно-цифровых символов без пробелов или специальных символов (,);.

Списки в Лиспе - основное средство представления знаний. Например, с помощью вложенных списков может быть представлена характеристика человека:

```
(сотрудник
(имя Петр)
(отчество Петрович )
(фамилия Иванов)
( образование ( среднее (с 1969 по 1979))
( высшее ( ВГУ г.Воронеж (с 1979 по 1982)
( МГУ г. Москва (с 1982 по 1984)) ) ( специальность
```


(техническая кибернетика)
(программирование)
(стаж (с 1984 по 1997)
)

8.3. ФУНКЦИИ

Функции в Лиспе аналогично математическим функциям ставят в соответствие элементам из одного множества - определения (аргументов) - единственный элемент из множества значений. В программах следует различать определение функций и вызов (применение) функций.

В языке Лисп принята единообразная префиксная форма записи, при которой как имя функции или действия, так и аргументы записываются внутри скобок:

(f x)
(g x y) (сумма_квадратов 2 3).

Аналогично записываются и арифметические действия:

(+ x y)
(*x(+yz))
(+ (^ x x) (^ y y)).

Определение функций и их вычисление в Лиспе основано на **лямбда-исчислении Черча**. В 1-исчислении Черча функция записывается в виде

1(x1,x2,...,xn) .fn

В Лиспе 1-выражение имеет вид (LAMBDA (x1, x2,..., xn).fn).

Символ LAMBDA означает, что мы имеем дело с определением функции. Символы x_i являются формальными параметрами, они образуют список, называемый лямбда-списком; fn - это тело функции, которая может иметь произвольную форму, допускаемую интерпретатором Лиспа. Телом функции может быть, например, константа или композиция из вызовов функций. Функцию, вычисляющую сумму квадратов двух чисел, можно, например, определить так:

(lambda(xy)(+(*xx)(*yy))).

Лямбда-выражение - это безымянная функция, которая может быть использована для связывания формальных и фактических параметров на время вычислений. Вызов такой функции происходит по форме

(лямбда-выражение a1 a2 ... an)

Здесь a_i - фактические параметры, с которыми происходит вычисление.
Например

((lambda (x y) (+ (* x x) (* y y))) 3 4).
Результат: 25.

Определить новую функцию и дать ей имя для последующих вызовов можно с помощью функции DEFUN (define function):

(DEFUN имя лямбда-список тело).

DEFUN соединяет символ с лямбда-выражением, и символ начинает представлять (именовать) определенные этим лямбда-выражением вычисления. Значением этой формы является имя

новой функции:

```
(defun sumsquare (x y) (+ (* x x) (* y y))) .
```

Результат: sumsquare.

Вызов (применение) этой функции:

```
(sumsquare 34)
```

Результат: 25.

Определение функции задается списком, поэтому его можно модифицировать в ходе выполнения программы. Кроме того, некоторый символ может быть и именем функции и переменной.

В Лиспе передача параметров происходит по значению. Формальные параметры функций являются статическими и локальными, т.е. действительны только внутри той формы, в которой они определены.

Основу для построения различных функций образует набор небольшого числа примитивных встроенных функций. Базовыми функциями обработки S-выражений являются функции

CAR, CDR, CONS, ATOM, EQ, EQL, =
и другие, смысл которых отражен в табл. 3.7.

Таблица 3.7
Базовые функции обработки S-выражений

Функ- ция	Вызов	Действие	Пример использования
CAR	(CAR спи-	Возвращает головную часть	(CAR(1 234))
CDR	(CDR спи-	Возвращает хвостовую часть	Результат:1 (CDR(! 234))
CONS	(CONS S- жение спи-	Возвращает все, кроме 1-го элемента	Результат:(2 3 4)
ATOM	(ATOMS- жение)	Строит список из переданных в качестве аргументов головы и	(CONS I (2 3 4))
EQ	(EQ символ символ)	Предикат; проверяет, является ли аргумент атомом, и возвращает (истина), либо Nil или ("ложь")	Результат: (1234) (ATOM A) : t (ATOM (1 2 3)): Nil
EQL	(EQL число число)	Предикат; проверяет тождество символов-аргументов, неприменяемо для чисел	(EQ A A): t (EQ X (CAR (X Y Z))): t
=	(= число число)	Предикат, проверяет тождество чисел одного типа	(EQL 3.0 3.0): t
EQUA	(EQUAL или список число или	Предикат, проверяет тождество чисел различных типов	(=30.3el):t
EQUA	(EQUALP объект объ-	Аналогична EQL, но, кроме того, проверяет идентичность списков	(EQUAL(xyz)(xyz)):t
NULL	(NULL спи-	Проверка наиболее общего равенства объектов	
NOT	(NOT логи- величина)	Проверка, является ли аргумент пустым списком	
NTH	(NTH n спи-	Логическое отрицание	
FIRST		Выделение n-го элемента списка	(NTH 2 (1 2 3)): 3 (индексы начинаются с 0)
SEC-		Предикаты, выделяющие	
LAST		Соответствующие элементы	

Отметим, что в программах на Лиспе надо тщательно отличать значения от их обозначений.

В Лиспе константы обозначают самих себя. Выражения типа `(* 2 2)` сразу вычисляются. Чтобы избежать нежелательного вычисления выражения используется функция `QUOTE` или знак апострофа (`'`) перед выражением:

`(* 2 2) : 4`

`'(* 2 2) : '(* 2 2)` – список

Произвольный символ можно использовать как переменную, и он может обозначать произвольное выражение. При первом использовании символу должно быть присвоено или с ним связано некоторое значение с помощью функции `SET`, например,

`(SET 'операции' (+ - * /))`

Знак `'` используется для подавления вычисления аргументов функции `SET`. Функция `SETQ` не вычисляет значения 1-го аргумента (а 2-го вычисляет).

На значение символа можно сослаться, указав его без апострофа (`'`).

Для занесения значений в ячейку памяти, связанной с символом, можно пользоваться обобщенной функцией присваивания `SETF`, размещающей значения в соответствующей ячейке памяти:

`(SETF ячейка_памяти значение).`

Переменная «ячейка_памяти» без апострофа указывает на ячейку памяти. Присвоение, выполняемое функциями `SET`, `SETQ` и `SETF`, является побочным эффектом, этих функций, помимо того, данные функции возвращают присваиваемые значения.

8.4. ФОРМЫ. УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ В ЛИСП-ПРОГРАММЕ

Программа состоит не только из функций, но и из форм. Простейшими формами являются константы, переменные, лямбда-вызовы, вызовы функций.

Остановимся более подробно на специальных формах, предназначенных для управления обработкой программы и контекстом. У каждой формы определен синтаксис и семантика, основанные на едином способе записи и интерпретации.

Управляющие предложения Лиспа внешне выглядят как вызовы функций - в виде скобочных выражений, первый элемент которых действует как имя управляющей структуры, а остальные элементы - как аргументы. Наиболее важные формы можно разделить на следующие группы:

работа с контекстом

- `QUOTE` или блокировка вычисления,
- вызов функции и лямбда-вызов,
- предложения `LET` и `LET*`;

последовательное исполнение

- предложения `PROG1`, `PROG2` и `PROGN`;

разветвление исполнения

- условные предложения `COND`, `IF`, `WHEN`, `UNLESS`,
- выбирающее предложение `CASE`;

итерации

- циклические предложения `DO`, `DO*`, `LOOP`, `DOTIMES`, `DOUNTIL`;

передачи управления

- предложения `PROG`, `GO` и `RETURN`;

динамическое управление вычислением

- THROW, CATCH, а также BLOCK.

Эти управляющие формы (кроме QUOTE и лямбда-вызова, а также вызовов функций), в основном, используются в теле лямбда-выражений, определяющих функции.

Предложение LET используется для создания связи переменных внутри формы:

(LET ((пер1 знач1) (пер2 знач2)...) форма1 форма2 ...).

При вычислении этого выражения статические переменные пер1, пер2, ... связываются (одновременно) с соответствующими значениями знач1, знач2, ..., а затем вычисляются значения форм форма1, форма2, ... Значение последней формы возвращается как общий результат. Форма LET* отличается от LET лишь тем, что связывание переменных и вычисление форм происходит не одновременно, а последовательно, вначале 1-е, потом 2-е и т.д.

Например:

```
(let*((x2)(y(*3x)))
```

```
(list x y)
```

Результат: (2 6).

Предложения PROG1, PROG2 и PROGN позволяют организовывать последовательные вычисления из нескольких вычисляемых форм:

```
(PROG1 форма1 форма2 ... форман)
```

```
(PROG2 форма1 форма2 .. форман)
```

```
(PROGN форма1 форма2 . форман).
```

Различие этих форм лишь в возвращаемых ими в качестве общего значения результатах. Форма PROG1 возвращает значение формы1, PROG2-формы2, PROGN -последней формы n.

Например:

```
(progn (setq x 2) (setq y (* 3 x)))
```

Результат: 6.

Предложение COND является основным средством разветвления обработки. Структура условного предложения такова:

(COND (p1 a1) (p2 a2)... (pn an)).

p_i - это предикаты (выражения-условия, которые могут быть либо истинными (T), либо ложными (NIL)). Их значения вычисляются слева направо, пока не будет получено значение «истина» (T), затем вычисляется и возвращается в качестве результата результирующее выражение a_i, соответствующее 1-му истинному предикату p_i. Если истинного предиката нет, то значение COND - NIL. Форма a_i для соответствующего предиката может отсутствовать (тогда возвращается значение этого предиката в случае его истинности), или, наоборот, может быть задана последовательность форм для предиката p_i - тогда эти формы вычисляются последовательно и возвращается значение последней.

В следующем примере с помощью предложения COND определена функция, устанавливающая тип выражения:

```
(defun тип (1)
```

```
(cond ((null 1) 'пусто)
```

```
((atom 1) 'атом)
```

```
(t 'список)))
```

Результат: ТИП.

Примеры применения этой функции:

(тип ' (a b c))
Результат: СПИСОК.
(тип (atom ' (a t o m)))
Результат: ПУСТО.

Для организации ветвления можно использовать и формулы IF, WHEN, UNLESS:

(IF условие то-форма иначе-форма),

что эквивалентно

(COND (условие то-форма) (Т иначе форма));
(WHEN условие форма1 форма2 ...),

что эквивалентно

(UNLESS (NOT условие) форма! форма2 ...)

или

(COND (условие форма1 форма2 ...)).

Можно применять и выбирающее предложение CASE:

(CASE ключ (список ключей1 форма11 форма12 ...)
(список ключей2 форма21 форма22 . . .)

В этой форме сначала вычисляется значение ключевой формы «ключ», затем происходит сравнение с элементами списков ключей и, если найдено значение ключевой формы, вычисляется последовательность соответствующих форм, значение последней из которых возвращается как значение всего выражения CASE.

Предложения PROG, GO и RETURN аналогичны конструкциям неструктурных языков программирования (типа FORTRAN, Бейсик); пользоваться ими не рекомендуется.

8.5. РЕКУРСИЯ И ЦИКЛ В ПРОГРАММАХ НА ЛИСПЕ

В «чистом» функциональном программировании организация повторяющихся вычислений должна происходить лишь с помощью условных предложений и определения рекурсивных, вызывающих самих себя, функций. Рассмотрим в качестве примера функцию, просто определяемую через рекурсию, - факториал $n! = 1 * 2 * 3 * \dots * (n-1) * n = (n-1)! * n$ ($0! = 1$ по определению):

(defun ! (n) (if(= n 0) 1 (* n (! (. n 1))))) .

Имя функции - "!", ее аргументом является переменная n. Лямбда-выражение, определяющее функцию, представляет собой условную if-форму, которая в случае $n=0$ возвращает 1, а в противном случае вычисляет произведение n и результата вызова этой же функции ! для аргумента n-1.

Пример вызова этой функции:

(!5)
Результат: 120.

В случае повторяющихся вычислений в Лиспе могут быть использованы не только рекурсивные функции, но и известные по процедурным языкам циклы. Самым общим циклическим

предложением в Лиспе является DO, имеющее следующую форму:

```
(DO ((пер1 знач1 шаг1) (пер2 знач2 шаг2) ...)  
(условие-окончания форма11 форма12 ...)  
форма21 форма22 ...)
```

Вычисление предложения DO начинается с присваивания переменным пер1, пер2, ... начальных значений знач1, знач2, . . . соответственно; потом вычисляется условие окончания и, если оно истинно, последовательно вычисляются формы форма1i, и значение последней возвращается как результат DO-предложения. В противном случае вычисляются формы форма2i из тела предложения DO, затем значения переменных пер1, пер2, . . . изменяются на величину шага шаг1, шаг2, ... и все повторяется.

Для примера с помощью предложения DO определим функцию expt, вычисляющую n-ю степень числа x (n - целое положительное):

```
(defun expt (x n)  
  (do ((результат 1) ; начальное значение  
      ((= n 0) результат) ; условие окончания  
      (setq результат (* результат x))  
      (setqn(^nl))))
```

Результат задания функции: EXPT.

Пример вызова:

```
(expt 2 3)
```

Результат: 8.

Итеративные (циклические) и рекурсивные программы теоретически одинаковы по своим вычислительным возможностям, однако свойства итеративных и рекурсивных вариантов программ могут существенно различаться. Рекурсивные программы более короткие и содержательные. Особенно полезно использовать рекурсию в тех случаях, когда решаемая задача и обрабатываемые данные по своей сути рекурсивны, например, при обработке списков, так как списки могут рекурсивно содержать подсписки, при работе с другими динамическими структурами, которые заранее не полностью известны. Рекурсивные процедуры играют важнейшую роль почти во всех программах, связанных с искусственным интеллектом.

8.6. ВВОД-ВЫВОД ДАННЫХ

До сих пор рассматривался ввод и вывод данных в лисповских программах через параметры функций и свободные переменные. Для организации диалога человека с программой в Лиспе существуют специальные функции READ и PRINT.

Для вывода результатов можно использовать функцию PRINT. Это функция с одним аргументом, которая сначала вычисляет значение аргумента, а затем выводит это значение.

Например:

```
(PRINT (* 2 2))
```

Результат: 4.

Перед выводом происходит переход на новую строку.

Функция READ читает и возвращает выражение: (READ). Как только интерпретатор встречает такое предложение, вычисления приостанавливаются до тех пор, пока не будет введен какой-либо символ или целиком выражение. Аргументов у функции READ нет, ее использование построено на побочном эффекте, состоящем именно во вводе выражения. Прочитанное выражение можно сохранить для следующего использования и обработки, например, так:

```
(setq input (read));
```

прочитанное READ выражение присваивается переменной input.

Лисповские операторы ввода-вывода очень гибки, их можно использовать в качестве аргументов других функций. Для более эстетичного оформления вывода можно использовать функции PRINC, печатающую строку без окаймляющих кавычек и со специальными символами, а также TERPRI, переводящую строку.

Для форматного вывода (в соответствии с некоторым образом) существует функция FORMAT, обладающая гибкими возможностями, описанными в руководствах по языку Лисп.

Помимо стандартных устройств ввода-вывода, может осуществляться обработка файлов на магнитных носителях, загружаться из файлов определения функций и т.д.

8.7. ПРИМЕР ПРОГРАММИРОВАНИЯ НА ЛИСПЕ

Рассмотрим в качестве примера программирования на Лиспе менее элементарную классическую задачу, носящую название игры в «ханойские башни».

Игра состоит в следующем. Используются три вертикальных стержня A, B, C и набор N дисков разного диаметра с отверстием посередине (так что их можно надевать на стержни). В начальном положении все диски надеты на стержень A по порядку убывания диаметров: внизу самый большой, над ним - поменьше и т.д., а наверху - самый маленький. Целью является перенос всех дисков со стержня A на стержень B по следующим правилам:

- 1) за один раз можно перенести только один диск;
- 2) больший по размеру диск нельзя положить на меньший;
- 3) третий стержень C можно использовать как вспомогательный. Алгоритм решения задачи

можно представить в виде трех следующих рекурсивных подзадач:

- 1) перенести со стержня A N-1 дисков на вспомогательный стержень C;
- 5) перенести нижний диск со стержня A на стержень B;
- 6) перенести со стержня C N-1 дисков на стержень B.

Программа состоит из трех последовательно определяемых функций «ханойские-башни», «перенос», «выведи» и имеет вид:

Программа 130

```
(defun ханойские-башни (высота)
  (progn (перенос "a" "b" "c" высота) "готово"))
ХАНОЙСКИЕ-БАШНИ
(defun перенос (из в вспомогательный n)
  (cond
    ((= n 1) ; ветвь 2
     (выведи из в) (t (перенос из ; ветвь 1
                          вспомогательный
                          в
                          (- n 1))
                     (выведи из в)
                     (перенос вспомогательный ; ветвь 3
                              в
                              из
                              (- n 1))))))
ПЕРЕНОС
(defun выведи (из в)
  (format t "~S -> ~S~%" из в))
ВЫВЕДИ
```

Вызов функции «ханойские башни» дает такое решение:

(ханойские-башни 3)

A->B

A->C

B->C

A->B
C->A
C->B
A->B
ГОТОВО

Можно убедиться, что определенная нами функция дает правильное решение для произвольного числа дисков, однако время решения задачи с увеличением числа дисков быстро возрастает.

8.8. СВОЙСТВА СИМВОЛОВ

В Лиспе могут быть определены, так называемые, свойства символов. Список свойств имеет вид:

(имя_свойства1 значение1 имя_свойства2 значение2 . . . имя_свойстваN значениеN).

Присваивание нового свойства или изменение значения существующего осуществляется с помощью функции PUTPROP (или просто PUT):

(PUTPROP символ свойство значение).

Выяснить значение свойства, связанного с символом, можно с помощью функции GET:

(GET символ свойство).

С использованием этой функции можно также присваивать свойства символам:

(SETF (GET символ свойство) значение).

Свойства символов глобальны. Эта конструкция языка Лисп полезна во многих типичных случаях представления данных, в том числе семантических сетей, фреймов и объектов объектно-ориентированного программирования.

Контрольные вопросы и задания

1. В чем состоит основная идея функционального программирования?
2. Что называется символом в программировании на Лиспе?
3. Что такое атомы в программах на Лиспе?
4. Что такое список?
5. Охарактеризуйте примитивные функции языка Лисп.
6. Как можно связать с символом некоторое значение? Как поместить значение в ячейку памяти?
7. Приведите примеры 1-выражений в Лиспе.
8. Как можно определить функцию и дать ей имя для последующих вызовов в Лиспе?
9. Охарактеризуйте управляющие формы в Лиспе.
10. Какую роль играет в функциональном программировании рекурсия?
11. Запишите рекурсивные определения функции проверяющую наличие в списке некоторого заданного элемента, подсчитывающую число элементов в списке, соединяющую два списка (с использованием точечной нотации).

§9. ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

9.1. ОСНОВНЫЕ ПОЛОЖЕНИЯ

Как уже отмечалось выше (п. 4.1), в настоящее время растет популярность методологий, ориентированных на данные. В первую очередь, это объектно-ориентированное программирование.

Объектно-ориентированная методология проектирования программ основана на концепциях упорядочивания информации и абстрактных типов данных. Такой подход рассматривает все такие ресурсы как данные, модули и системы в качестве объектов. Каждый *объект* содержит некоторую структуру данных (или тип данных), обрамленную набором процедур (методов), предназначенных для манипулирования этими данными. Используя эту методологию, программист может создать свой собственный абстрактный тип и отобразить проблемную область в эти созданные им абстракции вместо традиционного отображения проблемной области в предопределенные управляющие структуры и структуры данных языка программирования. Подобный подход является более естественным, чем методологии, ориентированные на обработку (на процесс), из-за возможности использовать в процессе программирования разнообразные виды абстракции типов данных. На этом пути программист может сконцентрироваться на проекте системы, не беспокоясь о деталях информационных объектов, используемых в системе.

Основные шаги разработки программы, предусмотренные данной методологией:

- определить проблему;
- развить неформальную стратегию, представляющую общую последовательность шагов, удовлетворяющую требованиям к будущей программе;
- формализовать стратегию
- идентифицировать объекты и их атрибуты;
- идентифицировать операции;
- установить интерфейсы;
- реализовать операции.

Большинство современных языков и систем программирования развивается в направлении все большего использования объектной методологии в создании программ. Характерными примерами являются универсальные языки Паскаль, СИ и даже Бейсик, в современных версиях которых появились средства объектно-ориентированного программирования. Так, начиная с версии 5.5, Турбо-Паскаль охватывает метод проектирования программ на основе объектно-ориентированного программирования.

9.2. ОСНОВЫ ОБЪЕКТНОГО ПРОГРАММИРОВАНИЯ В СИСТЕМЕ ТУРБО-ПАСКАЛЬ

Объект в ТурбоПаскале - это структура данных, содержащая поля данных различных типов и заголовки методов и обобщающая структуру «Запись» (record).

Синтаксис описания объекта:

```
<ИмяПотомка>=
    object<ИмяПредка> поле;

        поле;

        ...
        метод;

        ...
        метод;

end;
```

В отличие от записи полями объекта могут быть, кроме данных, еще и методы, обрабатывающие эти данные

Метод - это процедура или функция, объявленные внутри описания объекта. Синтаксис описания метода:

```
procedure <Заголовок>(<Параметр1>, <Параметр2>:integer),
```

Метод имеет доступ к полям данных объекта, не требуя передачи их ему в виде параметров

Объявление метода внутри объявления объектного типа содержит только заголовок. Тело метода определяется вне объявления объекта. Его заголовок должен содержать имя объекта, которому принадлежит метод. Например:

```
procedure <ТипОбъекта.Метод>
  (<Параметр1>, <Параметр2> : integer);
begin
  ...
  ...
end; {Метод}
```

Методы подразделяют на статические и виртуальные. Виртуальный метод отличается от статического тем, что реализующий его код подсоединяется к исполняемой программе не в процессе компиляции, а в процессе выполнения, что достигается, так называемым, поздним связыванием. Это дает возможность строить иерархию объектов с одинаковыми названиями методов, реализуемыми, однако, различными кодами.

Синтаксис виртуального метода:

```
procedure <Метод> (<Параметр1>, <Параметр2> : integer): virtual;
```

Кроме обычных процедур и функций, ТурбоПаскаль 6.0 реализует два специальных типа методов: конструктор и деструктор.

Конструктор - это специальный метод, инициализирующий объект, содержащий виртуальные методы, он объявляется специально зарезервированным словом `constructor`. Конструктор инициализирует объект путем установления связи между объектом и специальной таблицей виртуальных методов, содержащей адреса кодов, реализующих виртуальные методы. Конструктор может также использоваться для инициализации полей данных объекта.

Деструктор - это специальный метод, освобождающий память «кучи» от динамических объектов. Он объявляется с использованием специально зарезервированного слова `destructor`.

Основными отличительными свойствами объекта являются

- инкапсуляция - объединение записей с процедурами и функциями, работающими с этими записями;
- наследование - задание объекта, затем использование его для построения иерархии порожденных объектов с наследованием доступа каждого из порожденных объектов к коду и данным предка;
- полиморфизм - задание одного имени действию, которое передается вверх и вниз по иерархии объектов с реализацией этого действия способом, соответствующим каждому объекту в иерархии.

Рассмотрим смысл каждого из перечисленных свойств на примере построения на экране дисплея точек разных цветов (звездного неба).

Инкапсуляция. Основой решения задачи является задание положения (позиции) отдельной точки на экране, описываемого координатами X и Y . Для задания координат подходит тип «запись»:

```
Pozition = record
  X, Y : integer;
end;
```

Далее может быть необходимо задать значения координат (такая процедура носит название инициализации). Создадим соответствующую процедуру:

```
procedure Init(CoordX, CoordY : integer);
begin
  X := CoordX;
```

```
Y := CoordY;
end;
```

Затем потребуется знание фактических значений координат. Для этого вводим две функции:

```
function GetX : integer;
begin
  GetX := X;
end;
function GetY : integer;
begin
  GetY := Y;
end;
```

По нашему замыслу процедура Init и функции GetX и GetY должны работать только с полями записи Pozition.

Введение объектов позволяет зафиксировать это положение, объявив и поля, и действия над ними в единой конструкции:

```
Pozition = object
  X, Y: integer;
  procedure Init(CoordX, CoordY : integer);
    function GetX : integer;
    function GetY : integer;
  end;
```

Процедура Init и функции GetX и GetY являются методами объекта Pozition.

Для инициализации экземпляра типа Pozition достаточно вызвать его метод, как если бы он был полем записи:

```
var
  FirstPozition : Pozition;
  ...
  FirstPozition.Init(10,15);
```

Метод задается так же, как и процедура в модуле: внутри объекта записывается заголовок (как в секции Interface модуля); при этом все поля, используемые методом, должны предшествовать его объявлению. Определение метода (расшифровка действий) происходит вне объявления объекта. Имя метода должно предваряться названием типа объекта, которому метод принадлежит, сопровождаемым точкой. Например,

```
procedure Pozition. Init(CoordX, CoordY: integer);
begin
  X := CoordX;
  Y := CoordY;
end;
```

Заметим, что имена формальных параметров метода не могут совпадать с именами полей данных объекта.

Также как модуль скрывает детали реализации процедур от пользователя, объект может скрывать свои поля и методы. Для этого используется ключевое слово private (личный). Личные поля и методы доступны только внутри метода. Объявление выглядит следующим образом:

```
type
  ObjectName=object
    поле;
```

```

...
поле;
метод;

метод;
private
поле;
...

поле;
метод;
...

метод;
end;

```

Наследование. Рассмотрим точку с координатами X и Y. Ее можно сделать видимой или невидимой, ей можно задать цвет, ее можно переместить. Создадим объект с такими возможностями:

```

Point=object
  X,Y : integer; "
  procedure Init(CoordX, CoordY : integer);
  function GetX : integer;
  function GetY : integer;
  Visible: Boolean;
  Color: Word;
  procedure Init(CoordX, CoordY : integer; InitColor : Word);
  function Is Visible : Boolean;
  procedure Show; {показывает точку}
  procedure Blind; {стирает точку}
  procedure Jump(NextX, NextY : integer); {перемещает точку}
end;

```

Заметим, однако, что поля X,Y и методы GetX, GetY практически совпадают с соответствующими полями и методами объекта Position.

Турбо-Паскаль предоставляет возможность учесть эту ситуацию. Следует считать тип объекта Point порожденным типом Position, записав это следующим образом:

```

Point=object(Position)
  Visible : Boolean;
  Color : Word;
  procedure Init(CoordX, CoordY : integer; InitColor : Word);
  function Is Visible : Boolean;
  procedure Show;
  procedure Blind;
  procedure Jump(NextX, NextY : integer);
end;

```

Объект Point теперь наследует свойства объекта Position. Поля X,Y явно не заданы в Point, но Point ими обладает благодаря наследованию, т.е. можно написать

```
Point.X:=17;
```

Смысл объектно-ориентированного программирования заключается в работе с полями объекта через его методы.

Полиморфизм. Предположим, надо создать объект «кружок». Очевидно, что новый объект должен иметь предком объект Point, обладая всеми его свойствами, кроме того, быть больше по размеру. Однако, ясно, что «высветить» точку и закрашенную окружность не удастся одними и теми же командами. Турбо-Паскаль разрешает сохранить потомку имя родительского метода, «перекрывая» его. Чтобы перекрыть родительский метод, надо просто задать его с тем же именем, но с другим телом (кодом) и, если необходимо, с другим набором параметров. Такой метод делается виртуальным и к его объявлению добавляется слово virtual. Применение виртуальных методов налагает ограничения на процедуры инициализации, которые должны записываться с зарезервированным словом constructor и иметь общее имя Init

Каждый отдельный экземпляр объекта должен инициализироваться с помощью отдельного вызова конструктора

Для очистки и удаления динамически распределенных объектов существует специальная процедура destructorDone.

Деструктор комбинирует шаг освобождения памяти в «куче» с некоторыми другими задачами. Метод деструктора может быть пустым, поскольку работу выполняет не только код тела, но и код, генерируемый Турбо-Паскалем в ответ на зарезервированное слово destructor.

9.3. ОБОЛОЧКА TURBO-VISION

Объектно-ориентированный подход позволяет создавать объектные среды, библиотеки объектов для последующего использования при разработке программ. Особенно широкую известность получили объектные оболочки для разработки интерфейсов программ в некотором едином, ставшем стандартным, стиле.

Turbo-Vision - это оболочка для создания оконченных интерфейсов программ в среде Турбо-Паскаль версий 6.0, 7.0.

Объектно-ориентированная библиотека Turbo-Vision включает

- многократные перекрещивающиеся окна с изменяемыми размерами;
- выпадающие меню;
- поддержку мыши;
- встроенную установку цвета;
- кнопки, полосы скроллинга, окна ввода, зависимые и независимые кнопки;
- стандартную обработку клавиш и нажатий мышки и многое другое.

Стандартный отработанный интерфейс, создаваемый с помощью Turbo-Vision, имеет сверху строку меню, а снизу - строку статуса.

Текст, который появляется в строке статуса, обычно выводит сообщения о текущем состоянии программы, показывая доступные горячие клавиши или подсказки для команд, которые доступны пользователю в данный момент.

Когда «выпадает» меню, полоса подсветки пробегает по списку элементов меню при движении мыши или нажатии на клавиши курсора. Нажатием клавиши <Enter> или левой кнопки мыши выбирается подсвеченный элемент. Выбранный элемент меню посылает команду в определенную часть программы.

Программа обычно взаимодействует с пользователем через одно или более окон или диалоговых окон, которые появляются и исчезают на панели экрана в соответствии с командами от мышки или клавиатуры. Turbo-Vision предоставляет большой набор окон для ввода и отображения информации. Окна можно сделать со скроллингом, что позволяет выводить в окнах большие объемы данных. Скроллинг окна по информации выполняется передвижением по полосе скроллинга внизу окна, с правой стороны окна или обеим сторонам. Полоса скроллинга указывает относительную позицию окна в отображаемых данных.

Диалоговые окна часто содержат кнопки - подсвеченные слова - которые могут активизироваться с помощью мыши (или переходом через клавишу Tab и нажатием клавиши <Пробел>). Отображаемые слова, выбранные мышью, могут быть установлены для передачи команд в программу.

Простейшая программа на Turbo-Vision выполняет следующие действия:

- заполняет панель экрана полутеневым шаблоном;
- выводит на экран полосу меню (наверху) и строку статуса (внизу);

- устанавливает обработчик для событий, инициированных от клавиш и мыши;
- строит объект меню и соединяет его с полосой меню;
- строит диалоговое окно;
- связывает диалоговое окно с меню;
- ожидает действий пользователя через мышь или клавиатуру. Краеугольным объектом любой программы является объект TApplication. TApplication - это абстрактный тип объекта. Он ничего не делает. TApplication используется для создания порожденных типов от TApplication, которые содержат программный код. Например, в простейшей демонстрационной программе HELLO.PAS, выводящей в стандартном для Turbo-Vision виде надпись HELLO, WORLD! определяется порожденный тип объекта THelloApp:

```
PHelloApp = ^HelloApp;
THelloApp = object(TApplication)
  procedure GreetingBox;
  procedure HadleEvent(var Event: TEvent); virtual;
  procedure InitMenuBar; virtual;
  procedure InitStatusLine; virtual;
end;
```

Как показано здесь, полезно определить тип указателя на каждый тип определяемого объекта, поскольку большая часть работы над объектами производится через указатели. Полиморфизм, главным образом, работает через указатели.

THelloApp содержит гораздо больше, чем четыре содержащихся в нем метода; порожденный объект наследует все от его предка. В определении THelloApp определяется, чем новый объект отличается от его предка TApplication. Все, что не переопределяется, наследуется в неизменном виде от TApplication.

Четыре метода, определенные в THelloApp, завершают приложения:

1) метод **HadleEvent** определяет, что некоторые события произошли (и указывает это) и как на эти события отвечать другим прикладным функциям; метод HadleEvent, определенный в TApplication, работает с общими событиями, которые возникают внутри любой программы, но необходимо обеспечить обработку событий, специфичных для программы;

2) метод **InitMenuBar** устанавливает меню для полосы Меню программы (TApplication содержит полосу меню, но не сами меню);

3) метод **InitStatusLine** устанавливает текст строки статуса внизу экрана; этот текст обычно отображает сообщение о текущем состоянии программы, показывая доступные горячие клавиши или напоминая пользователю о некоторых действиях;

4) метод **GreetingBox** вызывает диалоговое окно в ответ на элемент меню Greeting; GreetingBox вызывается из метода HadleEvent в ответ на событие, переключаемое выбором элемента меню Greeting; в более сложных программах можно использовать различные методы в ответ на каждый элемент меню, определяемый в начальном меню.

На верхнем уровне абстракции главная программа на Turbo-Vision выглядит аналогично Hello и содержит три метода:

```
var
  HelloWorld: THelloApp;
begin
  HelloWorld.Init;
  HelloWorld.Run;
  HelloWorld.Done;
end;
```

Каждый из этих трех методов требует небольшого пояснения.

Init - это необходимый вызов конструктора. Все объекты, содержащие виртуальные методы, должны быть созданы (через вызов их конструктора) до вызова любых других методов этого

объекта. Все конструкторы Turbo-Vision имеют имя Init. Это удобное соглашение, которым мы рекомендуем пользоваться.

HelloWorld.Init устанавливает объекты главной программы для использования. Он очищает экран, устанавливает некоторые значения для важных переменных, заполняет панель экрана и выводит строку статуса и полосу меню. Он вызывает конструкторы многих других объектов, некоторые из которых никогда не видны, поскольку они вызываются внутри.

Почти вся работа программы в Turbo-Vision выполняется внутри метода главной программы **Run**. Определения метода Run здесь нет, поскольку Run наследуется от родительского объекта TApplication.

Этот метод состоит, главным образом, из цикла repeat...until, показанном здесь в псевдокоде;

```
repeat
    <получить событие>;
    <обработать событие>;
until Quit;
```

Это не точный код, а общая схема реализации Run. В сущности программа Turbo-Vision циклически выполняет две задачи: получение события (где событие - это, в сущности, «сделать что-нибудь») и обработку этого события. Одно из событий вычисляется в некоторый вид команды выхода и цикл завершается.

Дестрактор **Done** в действительности очень прост: он уничтожает все объекты программы - полосу меню, строку статуса и панель экрана - и удаляет обработчик ошибок и драйверы Turbo-Vision. Метод Done должен отменять все действия, выполненные конструктором Init, а затем вызывать TApplication.Done, который обрабатывает все стандартные элементы.

Объектно-ориентированное программирование с использованием Turbo-Vision требует хорошего знания иерархии объектов и спецификаций их методов.

Помимо объектно-ориентированной оболочки Turbo-Vision для Турбо-Паскаля, существует аналогичная более развитая система для языка C++. Объектно-ориентированное программирование стало основой интерфейса системы Windows. Современные объектно-ориентированные системы типа Visual Basic, Delphi совершенно скрывают механизмы программирования и позволяют разрабатывать интерфейсы программ-приложений вообще без кодирования команд, пользуясь лишь средствами наглядного проектирования на экране дисплея.

9.4.* СРЕДА ОБЪЕКТНОГО ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ DELPHI

* Раздел о Delphi написан Е.А.Ереминым.

Созданный в 1970 г. Н. Виртом, язык программирования Паскаль не оставался чем-то неизменным. Хотя основная его часть сохранилась прежней, появился целый ряд усовершенствований и дополнений, которые были направлены на облегчение программирования задач все возрастающей сложности. Например, типа данных string не было в первоначальной версии языка, в то время как представить без него современный Паскаль практически невозможно.

Позднее были добавлены более сложные конструкции: Н. Вирт предложил структуру программы из модулей unit, для возможности реализации объектного программирования был введен тип object. В результате возможности Паскаля значительно возросли и он по-прежнему мог удовлетворить запросы любого самого изощренного системного программиста. Известна, например, высокая оценка языка, которую дал в одной из своих книг знаменитый американский программист Питер Нортона: «Лично я пользуюсь и языком Паскаль, и языком Си. Мои популярные сервисные программы «Утилиты Нортона» были написаны на Паскале, а затем переписаны на Си. Мне нравятся оба эти языка. Я считаю, что сам по себе Паскаль лучше, в нем меньше возможностей допустить ошибку...».

С появлением графической среды Windows положение существенно изменилось. Конечно, реализовать программу с современным оконным интерфейсом на Паскале было возможно, но для этого требовалось немалое мастерство и много времени.

Чтобы исправить это положение, в 1996 г. фирма Borland, известная своими разработками в

области реализации языков программирования, выпустила компилятор нового поколения Delphi. Прежде всего, это мощный компилятор языка Паскаль, дополненный рядом существенно новых возможностей для программирования в среде Windows. Но не только. Delphi - это система, имеющая интерфейс качественно нового типа, позволяющий при составлении текста программы видеть те графические объекты, для которых она пишется - так называемая, система визуального программирования.

Delphi является системой программирования очень высокого уровня. Она берет на себя значительную часть работы по управлению компьютером, что делает возможным в простых случаях обходиться без особых знаний о деталях ее работы.

В отличие от традиционных систем программирования, Delphi даже «сама» пишет значительную часть текста программы: описания объектов, заголовки процедур и многое другое. Программисту остается только вписать необходимые строчки, определяющие индивидуальное поведение программы, которые система не в состоянии предугадать. Но даже здесь Delphi во многих случаях сама указывает место, где надо разместить эти строки.

Вершиной автоматизации процесса программирования являются, так называемые, эксперты. Эксперт - это диалоговое окно, которое помогает пользователю описать, что он хочет видеть в своей программе. Например, *Эксперт проекта* спрашивает, необходимо ли вам системное меню и какого из предложенных типов ваш проект. Проанализировав введенные ответы, Delphi пишет код программы на Паскале. Отметим, что эксперты могут быть созданы самим пользователем.

Чтобы составить себе некоторое представление о работе в Delphi, проследим за реализацией в системе конкретной пробной программы, рекомендованной в качестве первого шага освоения Delphi в одном из руководств. Опустим для простоты описание несущественных сейчас деталей, касающихся вида экрана и объектов управления на нем. Сосредоточим свое внимание на сути процесса создания программы в среде Delphi.

Еще до того, как программист успел что-то сделать, система при запуске уже выполнила значительную работу. Во-первых, она создала для будущей программы стандартное окно (в Delphi его принято называть **формой**). Во-вторых, уже сгенерирован текст программы довольно значительной длины, который необходим для порождения этой формы на экране.

Теперь попробуем что-нибудь сделать сами. Выберем из предлагаемого широкого ассортимента стандартных объектов наиболее простой - кнопку. Найдем ее изображение в верхней части экрана (в Delphi ее называют **палитрой компонентов**) и щелкнем на нем мышкой. Тем самым мы дали системе знать, что нам нужна именно кнопка. Остается указать, в какое место формы поместить компонент. Подведем курсор «мыши» к требуемому месту и снова щелкнем - появится изображение кнопки. При необходимости с помощью мыши кнопку можно легко передвинуть или изменить ее размеры.

Заметим, что текст программы после появления кнопки увеличился - Delphi автоматически добавила необходимое описание.

Уже сейчас можно посмотреть на то, что получилось, в действии. Запустим программу и увидим на экране стандартное окно Windows, содержащее кнопку. В отличие от этапа проектирования, компоненты при выполнении программы «оживают»: если щелкнуть по этой кнопке мышкой, то будет видно, как она нажимается. (Правда, никакого эффекта пока нет, так как он еще не запрограммирован.)

Итак, мы уже запустили собственное Windows-приложение, хотя не набрали еще ни одной строчки текста с клавиатуры!

Теперь сделаем нашу кнопку действующей. Вернемся в режим проектирования и дважды щелкнем мышкой по кнопке. После этого Delphi вынесет на первый план текст программы и установит маркер в то место, где мы должны набирать текст, тем самым как бы призывая описать действия по нажатию на кнопку. Введем строку

```
Form1.Color := clAqua
```

Несколько слов по ее расшифровке. Свойство Color (цвет) объекта Form1 (нашей формы) получает значение clAqua (буквы cl указывают на цвет, а Aqua - это название одного из 16 стандартных цветов). Снова запустим программу и нажмем кнопку - форма поменяет свой цвет.

Описанный выше пример хорошо показывает стиль работы с Delphi, существенно отлича-

ющийся от традиционного стиля программирования. Конечно, реальные задачи посложнее, но работа протекает примерно так же.

На примере описанной выше программы видно, что многие компоненты Delphi имеют свое визуальное изображение. Замечательным достоинством системы является то, что размещение компонентов на экране, а также задание начальных значений их свойств (размера, цвета, вида бордюра и др.) Delphi позволяет осуществлять на этапе конструирования формы без написания какой-либо программы. Для этой цели предусмотрено специальное окно, называемое **инспектором объектов**, в котором перечислены все доступные в режиме проектирования свойства выделенного компонента и их текущие значения. Разумеется, любое из них при необходимости легко изменить, что немедленно скажется на внешнем виде объекта. Например, если в окне Инспектора объектов изменить цвет, то система тут же перекрасит компонент. Иными словами, можно до запуска программы видеть, как будет выглядеть на экране проектируемая форма.

Такой способ работы с объектами, имеющими графическое представление, принято называть **визуальным программированием**.

Визуализация процесса позволяет значительно быстрее увидеть результат своих усилий, делает его наглядным. Не последнюю роль при этом, по-видимому, играют эмоции и эстетические чувства - стремление красиво разместить объекты, подобрать их цвет и т.п. Замечено, что даже люди, которые не умеют рисовать, часто с интересом и удовольствием занимаются построением изображений из готовых элементов.

Какие еще перспективные черты заложены в систему Delphi?

Напомним, что Delphi работает в среде Windows и позволяет создавать для нее программные продукты. Первая (16-разрядная) версия Delphi прекрасно функционирует под любой разновидностью Windows, в то время как вторая (32-разрядная; более мощная, но достаточно хорошо совместимая с первой) требует более мощной операционной системы, например, Windows'95. Учитывая, что фирма «Microsoft» - главный разработчик системного программного обеспечения для персональных компьютеров - уделяет сейчас основное внимание именно развитию Windows, с этой точки зрения компилятор для Windows нельзя не признать перспективным.

Кроме того, Delphi является системой объектного программирования. Delphi позволяет не только использовать уже ставшие классическими объекты типа **object**, но и позволяет создавать новые, которые могут иметь графическое изображение и обладать свойствами стандартных элементов среды Windows. Такие объекты получили название **визуальных компонентов**; для их описания используется специально зарезервированное слово **class**. Помимо уже встречавшихся выше формы и кнопки, примерами визуальных компонентов могут служить меню, списки, поля редактирования, полосы прокрутки, таблицы и многое-многое другое. К Delphi прилагается целая библиотека стандартных визуальных компонентов - Visual Component Library. Очень важно, что библиотеку эту каждый пользователь может пополнить, добавляя в нее собственные компоненты.

Еще одной существенной чертой системы программирования Delphi является ее открытость - почти все имеющиеся в системе объекты реализованы на языке Паскаль и могут быть легко дополнены новыми. Например, если вас по какой-либо причине не устраивает стандартный редактор чисел, вы можете написать собственный и подключить его к системе.

Таким образом, среда Delphi содержит в себе все наиболее передовые черты системы программирования. Она является мощным и в то же время несложным в использовании инструментальным средством для создания приложений с современным интерфейсом (в том числе и обучающих программ). Из-за своего высокого уровня системы программирования типа Delphi даже получили специальное название - Среда быстрой разработки приложений, **RAD** (Rapid Application Development).

Если посмотреть на компоненты, входящие в стандартную библиотеку, то многие из них словно специально созданы для обучающих программ. Рассмотрим несколько примеров.

Возьмем радиокнопки - группу кнопок, названную так благодаря функциональному сходству с переключателями в радиоаппаратуре. Основной особенностью радиокнопок является то, что из всей группы в нажатом состоянии всегда находится только одна кнопка. В педагогической интерпретации это выглядит как наиболее примитивный метод опроса - выбор единственного правильного ответа из списка предложенных. Если же вам захочется иметь несколько правильных ответов на вопрос, возьмите другую разновидность кнопок - **Check Box** (обычно этот термин переводят как «кнопки с независимой фиксацией»). У таких кнопок состояние никак не зависит от

окружающих кнопок.

Для тех, кому традиционное тестирование кажется устаревшим, найдутся другие компоненты, например: **Image** - образ, изображение. Помимо возможности разместить картинку на экране, этот полезный компонент обладает способностью «чувствовать на себе» щелчок мышки, что позволяет элементарно реализовать контроль вопросов типа «Найдите и укажите на карте остров Мадагаскар». Кроме того, во все компоненты библиотеки Delphi заложена технология «**Drag and drop**»-«Перенеси и оставь». Благодаря ей можно располагать объекты на экране определенным образом при выполнении задания «Составьте схему из батареи, амперметра, выключателя и резистора».

Наконец, если в компьютере есть устройство считывания с компакт-дисков или хотя бы набор звуковых файлов, в Delphi предусмотрен специальный компонент -проигрыватель, позволяющий включать в урок музыку или звуки.

Важным достоинством Delphi как инструментальной среды является то, что ее компилятор делает автономные EXE-файлы. Они будут работать в среде Windows даже на той ЭВМ, на которой нет системы Delphi. Это свойство выгодно отличает Delphi от других аналогичных сред, например, от Microsoft Visual Basic.

Завершая краткое знакомство с системой Delphi, рассмотрим некоторые наиболее важные базовые понятия, лежащие в ее основе.

Все объекты в Delphi характеризуются свойствами. **Свойство** - это атрибут объекта, определяющий то, как объект выглядит или как он может себя вести. Например, свойства, определяющие внешний вид кнопки: color - ее цвет, left и top -координаты левого верхнего угла, height и width - высота и ширина. В качестве примера свойств, определяющих, как кнопка может себя вести, опишем свойство «enabled» (от английского давать возможность, разрешать, разблокировать, включать). Оно может принимать значения TRUE или FALSE - в зависимости от этого кнопка либо будет фиксировать на себе щелчок мыши, либо нет.

Отметим, что свойство является своеобразным обобщением понятия поля объекта **object**, так как помимо имени и типа дополнительно содержит способы записи и чтения значения поля. Например, при выполнении строки `Form1.Color := clAqua` происходит не просто запись значения, характеризующего цвет, в определенное место памяти, а вызывается метод `Form1.SetColor(clAqua)`. Этот метод не только производит присвоение, но и перекрашивает форму.

Еще одно важное понятие Delphi - это событие. Термин «событие» заимствован из Windows, его значение проще всего объяснить на примерах. В частности, событиями являются воздействия пользователя на клавиатуру или мышь (нажатие на клавишу или ее отпускание, движение мыши и т.п.). Кроме того, событием являются любые изменения состояния экрана: создание окна, изменение его размеров и многие другие (таблица с описанием полного списка событий занимает несколько страниц).

В ответ на любое событие в системе Windows передает управление обработчику события (event handler). **Обработчик события** - это программа, которая определяет реакцию объекта на это событие. Если пользователь не предусмотрел действия по данному событию, Windows обработает его сама; в частности, она может просто проигнорировать событие.

Благодаря такой идеологии программа на Delphi фактически представляет собой совокупность относительно самостоятельных обработчиков всевозможных событий.

Рассмотрим еще одно понятие - **метод**. Метод - это процедура или функция класса, определяющая поведение объекта. Метод приводит к выполнению определенной последовательности действий, часть из которых может быть связана с изменением внешнего вида объекта. Так, применение к форме `Form1` процедуры `Close` не просто удаляет ее из списка окон Windows, но и обеспечивает его исчезновение с экрана. Другой пример - метод `SetFocus` для формы переносит на нее фокус ввода, т.е. она становится активной и принимает набор с клавиатуры. Кроме того, метод выносит изображение активного окна на первый план и выделяет цветом его заголовок.

Отметим, что понятие метода в Delphi не отличается от введенного ранее понятия при описании Object Pascal.

9.8. СИСТЕМА ОБЪЕКТНОГО ПРОГРАММИРОВАНИЯ SMALLTALK

Большой интерес представляют языки, изначально созданные для объектно-

ориентированного программирования. Самым ярким примером является язык Smalltalk.

Система Smalltalk представляет собой совокупность взглядов, методов и средств проектирования и эксплуатации современных систем обработки информации, базирующихся на ряде новых концепций в понимании и модельном представлении вычислительного процесса. Авторами системы являются сотрудники исследовательской группы фирмы XEROX (США), работающие над этим проектом с 1970 г.

Smalltalk принадлежит к классу, так называемых, объектно-ориентированных языков программирования. Объект в Smalltalk содержит и процедурную, и понятийную части (объект=данные+процедуры), причем для представления понятийной части используется аппарат абстрактных типов данных.

При решении задач на объектно-ориентированном языке используется абстрактная модель обработки данных

«объект-класс-сообщение».

Эта модель носит фундаментальный характер и легко может применяться в любой предметной области.

Объекты и сообщения. Основой объектно-ориентированного программирования (ООП) является **объект**. В качестве объектов могут выступать абстрактные данные (числа, символы, файлы и т.д.) или сущности моделируемой предметной области и их взаимосвязь (черепашка, цветок, собака, врач, треугольник и т.п.). Объекты взаимодействуют друг с другом, посылая и принимая сообщения.

Объект обладает собственной памятью для хранения информации и набором **методов** - операций для манипулирования этой информацией. Для активизации метода объекту посылается сообщение, являющееся аналогом обращения к функции в традиционных языках программирования.

В каждом сообщении должны быть указаны

- адресат-объект, которому посылается сообщение;
- имя сообщения;
- объекты-параметры, которыми манипулирует метод (необязательно).

Программа в Smalltalk - это последовательность сообщений, посылаемых различным объектам. Простые сообщения делятся на три вида: унарные, бинарные, ключевые, табл.3.8.

Унарное сообщение представляет собой конструкцию следующего вида:

<объект-адресат> <сообщение>.

Например:

'строка'размер.

Бинарное сообщение имеет следующий вид:

<объект-адресат> <арифметико-логическая операция> <объект-параметр>.

Например:

2+3

Ключевое сообщение представляет собой конструкцию, допускающую несколько параметров.

Например:

книга автор: 'Толстой' название: 'Война и мир'.

В случае композиции двух или более сообщений их обработка выполняется в соответствии со следующими правилами:

Правило 1. Унарное сообщение имеет более высокий приоритет, чем бинарное, а бинарное - более высокий приоритет, чем ключевое сообщение.

Правило 2. В рамках одного типа все сообщения имеют одинаковый приоритет. Сложное выражение выполняется слева направо.

Правило 3. Скобки меняют порядок вычисления.

Таблица 3.8.
Некоторые примеры сообщений и их результатов

Сообщение	Описание	(Класс)
1+2 Бинарное сообщение	объект-адресат: имя сообщения: объект-параметр: результат:	1 (МалоеЦелое) + 2 (МалоеЦелое) 3 (МалоеЦелое)
" привет' вПозиции: 5 Ключевое сообщение	объект-адресат: имя сообщения: объект-параметр: результат:	' привет' (Строка) в Позиции: 5(МалоеЦелое) е (Символ)
5факториал Унарное сообщение	объект-адресат: имя сообщения: объект-параметр: результат:	5(МалоеЦелое) факториал ----- 120(БольшоеПоложительное Целое)
5между: 3 и:7 Ключевое сообщение	объект-адресат: имя сообщения: объект-параметр: результат:	5(МалоеЦелое) между: и: 3, 7 (МалоеЦелое) истина
1+2 Бинарное сообщение	объект-адресат: имя сообщения: объект-параметр: результат:	1 (МалоеЦелое) + 2 (МалоеЦелое) 3 (МалоеЦелое)
' привет' вПозиции: 5 Ключевое сообщение	объект-адресат: имя сообщения: объект-параметр: результат:	'привет' (Строка) в Позиции: 5(МалоеЦелое) е (Символ)
5факториал Унарное сообщение	объект-адресат: имя сообщения: объект-параметр; результат.	5(МалоеЦелое) факториал ----- 120(БольшоеПоложнтельноеЦелое)
5между: 3 и:7 Ключевое сообщение	объект-адресат: имя сообщения: объект-параметр: результат;	5(МалоеЦелое) между: и: 3, 7 (МалоеЦелое) истина

Классы объектов и методы. Объект обладает свойствами, поведением и состоянием. Ему можно присвоить имя. Объекты с одинаковыми свойствами и поведением объединены в классы. Каждый объект входит в один класс и называется **экземпляром класса**.

Объекты из одного класса распознают одни и те же сообщения и имеют одинаковую структуру собственной памяти. Объект имеет собственную память - **переменные экземпляра**, где хранится информация о его свойствах и состоянии. Доступ к указанной информации имеет только сам объект.

С каждым объектом связан набор (протокол) сообщений, которые он понимает. Всякому сообщению из протокола соответствует реализующая его процедура, называемая методом.

Метод определяет реакцию объекта на данное сообщение, т.е. его поведение. Он состоит из операций над своими переменными экземпляра и из посылок сообщений другим объектам. В конечном счете объект возвращает ответ на посланное ему сообщение. Поскольку все объекты одного класса обладают одинаковым набором методов, последние хранятся в одном месте - в самом классе.

У каждого класса существует два типа методов:

- методы класса (используются, когда необходимо произвести какие-либо действия с целым классом, например, добавить к классу новый экземпляр);
- методы экземпляра (сообщения к экземплярам данного класса).

Один из основных классов языка Smalltalk - класс «Величина». Экземплярами этого класса являются объекты, которые можно измерять, сравнивать, упорядочивать и вычислять. К этому классу, в частности, относятся символы, числа, дата и время. Эта группа классов имеет следующую иерархию.

Символ. Экземпляры класса «Символ» являются расширенным множеством символов во внутреннем коде со значениями от 0 до 255.

Дата. Экземпляры класса «Дата» представляют собой даты, такие как
ЯНВАРЬ 1, 1980.

Время. Экземпляры класса «Время» представляют собой время, такое как 10 или 12:15.

Число. Система поддерживает три вида числа:

- целое;
- вещественное с плавающей точкой (если есть сопроцессор);
- дробь - рациональное целое.

Символ обозначается с помощью знака \$, например, \$a,\$9,\$M.

В языке Smalltalk имеется 6 типов переменных:

- *экземплярные переменные*, существуют в течение всего жизненного цикла объекта экземпляра класса, в заголовке которого они описаны;
- *временные переменные*, описываются внутри метода и существуют только во время выполнения данного метода;
- *переменные класса*, декларируются в описании класса, доступны всем экземплярам данного класса;
- *глобальные переменные*, доступны всем экземплярам всех классов;
- *переменные пула*, декларируются в описании класса и доступны экземплярам некоторого подмножества классов;
- *псевдопеременные*, специально зарезервированные идентификаторы, указывающие на специальные объекты.

Для присвоения значения какой-либо переменной (исключая псевдопеременные) используется операция : := (присваивание).

Экземпляры объектов можно сравнивать друг с другом.

Класс «Объект» содержит методы сравнения: =, ~=. Все классы, являющиеся подклассами этого класса, наследуют эти методы. Помимо этого, каждый класс может определить для себя другие методы сравнения. Например, классы из группы «Величина» имеют еще несколько методов: > < <= >=. Аналогичные методы определены для класса «Строка».

Результат сравнения объектов - экземпляр класса «Логический». Этот класс имеет два подкласса: Истина и Ложь. Экземплярами этих классов являются псевдопеременные истина и ложь соответственно. Задание новых экземпляров этих классов приводит к ошибке.

Классы «Истина» и «Ложь» имеют следующий набор методов:

и: - истина, если адресат и параметр - «истина», в противном случае - «ложь» (существует аналогичное бинарное сообщение - &)

или: - ложь, если адресат и параметр - «ложь», в противном случае - «истина» (существует аналогичное бинарное сообщение - |).

На основе этих методов реализованы условные конструкции. В общем виде условная конструкция имеет вид:

<условие>

если условие истинно, выполнить <выражение1> если условие ложно,
выполнить <выражение2>.

На языке ООП эта конструкция записывается таким образом:

<объект-адресат класса Истина или Ложь>

если Истина: <объект-параметр1>

если Ложь: <объект-параметр2>.

В данном случае <объект-параметр> представляет собой блок сообщений.

Возможна конструкция

<условие> еслиИстина: <объект-параметр>,

а также

<условие> еслиЛожь: <объект-параметр>.

Блок сообщений - это группа сообщений, заключенная в квадратные скобки. Блоки - это объекты специального вида, вычисление которых (т.е. выполнение списка внутренних операторов) производится путем посылки к ним сообщения-значения.

Блок может иметь аргументы. В этом случае сообщение для его вычисления будет ключевым:

[блок] значение: <значение аргумента>

Программирование на SmallTalk. Язык Smalltalk обладает богатым набором циклических конструкций, отраженных в табл. 3.9.

Программирование на SmallTalk состоит не только в использовании predetermined в языке объектов и их методов, но и в задании новых методов и объектов.

Вначале рассмотрим создание новых методов для уже существующих классов. Перед созданием нового метода необходимо ответить на следующие вопросы:

1) что будет объектом-адресатом для создаваемого метода;

2) каким будет объект-параметр;

3) что будет результатом, возвращаемым новым методом. Таким образом определяется сообщение, обращающееся к новому методу. Затем можно описать и добавить в соответствующий класс сам создаваемый метод по форме

имя Сообщения

"комментарии"

| локальные переменные сообщения |

^ возвращаемый результат.

Таблица 3.9
Циклические конструкции языка Smalltalk

Запись	Содержание

<число> разПовторить: [блок сообщений]	Повторить заданное <число> раз
[блок условия] покаЛожь: [блок сообщений]	Пока условие ложно, выполняются сообщения
блок условия] покаИстина: [блок сообщений]	Пока условие истинно, выполняются сообщения
<число1> до: <число2> через: <шаг>	Выполнить блок сообщений, пока значение
выполнить:	<переменной>, изменяющее свое значение
[:<переменная> блок сообщений]	с заданным шагом, принадлежит
<объект> выполнить:	промежутку (число1, число2)
[:<переменная> блок сообщений]	Значение <переменной> присваивается
<объект> выбрать:	последовательно элементам <объект>
[:<переменная> <условие>]	Изменяет <объект>, удаляя элементы, не
<объект> исключить:	удовлетворяющие условию
[:<переменная> <условие>]	Изменяет <объект>, удаляя элементы,
<объект> собрать:	удовлетворяющие условию
[:<переменная> : сообщение]	Заменяет каждый элемент <объекта> на результат
	выполненного сообщения

В качестве примера рассмотрим определение метода поиска максимального из двух целых чисел.

Объектом-адресатом будет целое число, следовательно сам метод будет принадлежать к классу «Целое». Параметром будет второе целое число, а возвращаемым результатом - максимальное из этих чисел.

Текст метода «макс» может быть следующим (комментарии приведены в кавычках):

```
макс: экзЦелое "имяметода с параметром"
| максимум | "список локальных переменных"
сам>экзЦелое "псевдопеременная "сам" означает объект-адресат"
еслиИстина: [максимум := сам]
еслиЛожь : [максимум := экзЦелое].
^Максимум.
```

Наследование и полиморфизм. Создание новых объектов происходит по принципам наследования и полиморфизма. Сами объекты подчиняются отношению наследования, т.е. могут быть представлены в виде иерархической структуры с помощью дерева.

Чтобы создать объект, классу посылается сообщение о создании нового экземпляра. Класс создает экземпляр с присущей всем объектам этого класса структурой -набором переменных экземпляра. Он их инициализирует и, если в этом есть необходимость, выдает созданный экземпляр в качестве ответа на сообщение.

Например:

```
х:= Массив новыйЭкземпляр: 10.
```

(создается новый объект с именем х, класса Массив, размера 10).

Возможно создание нового экземпляра в результате выполнения некоторых сообщений.

Например:

```
' Привет,', 'мартышка'
(создается новый экземпляр класса «Строка», имеющий значение 'Привет, мартышка )
1/2
(объект-адресат и объект-параметр - экземпляры класса «Целое», а результат -экземпляр
класса «Дробь»)
```

```
1>2 (результат класса «Ложь»)
```

Каждый класс имеет одного предка, называемого **суперклассом**. Класс может иметь одного или нескольких потомков, называемых **подклассами**.

Класс «Объект» не имеет суперкласса и является корнем дерева иерархии классов.

Всякий класс наследует переменные экземпляра и методы своего суперкласса. Кроме того, он может содержать новые переменные экземпляра и методы, может переопределять у наследованные.

Знание иерархии классов важно для понимания процессов выдачи ответа на сообщение. При получении сообщения объект ищет в протоколе методов экземпляра своего класса метод с именем, совпадающим с именем сообщения. Если такого метода нет, он ищется в списке методов суперкласса и так далее. Когда метод обнаружен, он выполняется и выдается ответ. Если нигде, вплоть до корня дерева, метод не будет обнаружен, выдается сообщение об ошибке. Дерево предопределенных классов языка SmallTalk приведено ниже, рис. 3.18.

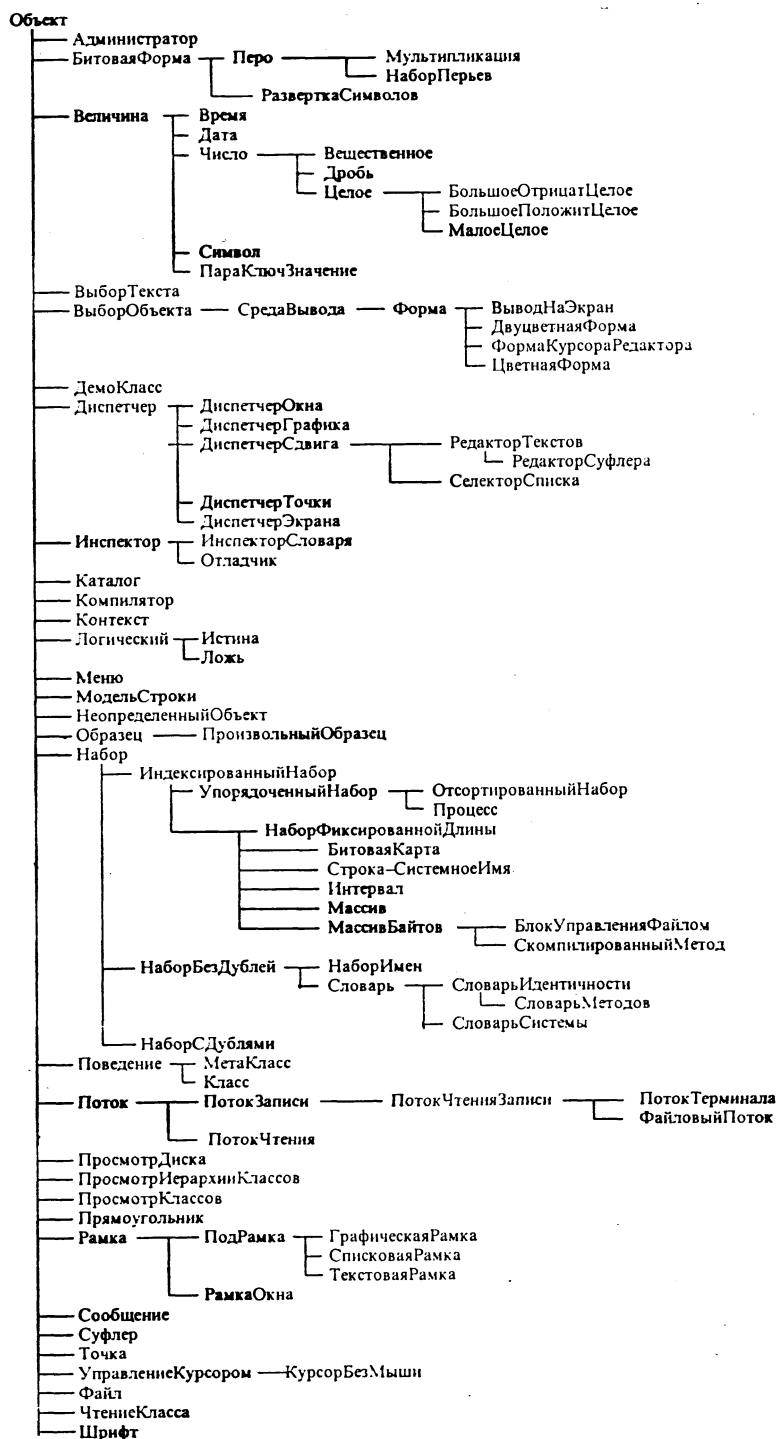


Рис. 3.18. Дерево предопределенных классов языка Smalltalk

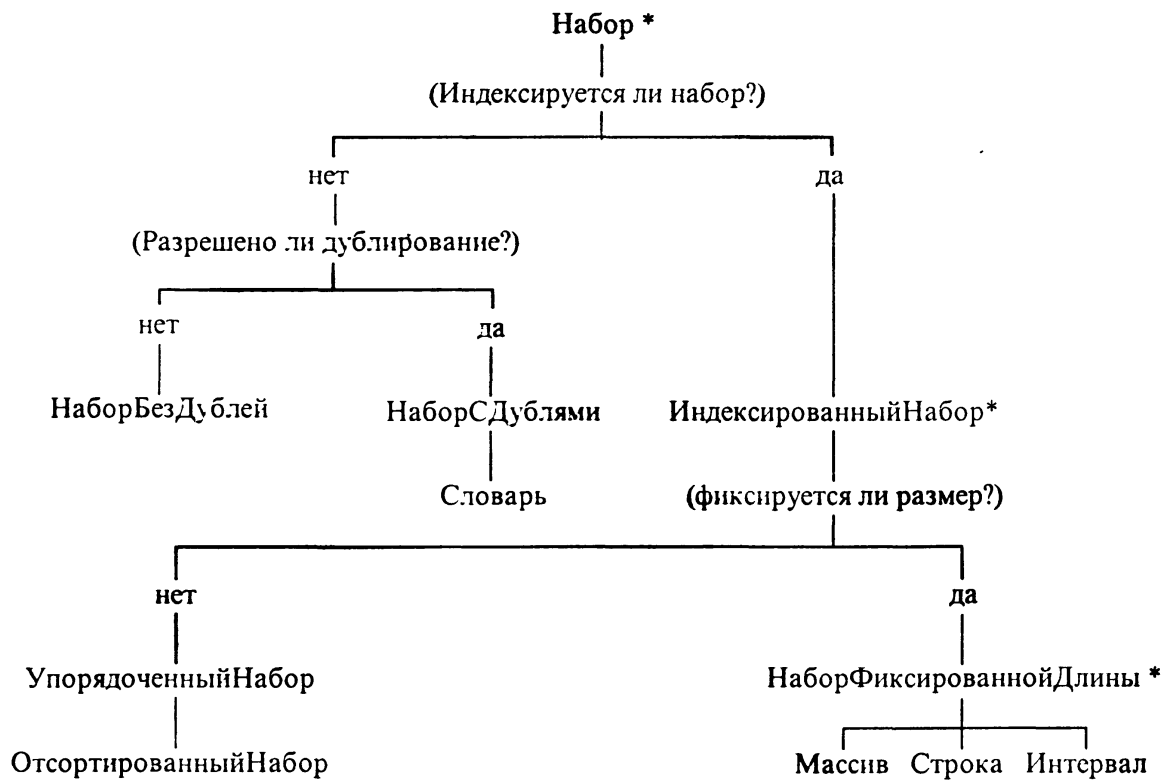


Рис. 3.19. Основные подклассы класса «Набор»
 (* - абстрактные классы, экземпляры которых не создаются)

В системе SmallTalk существует два типа объектов:

- 1) объекты с поименованными переменными (переменные с именами);
- 2) объекты с индексируемыми переменными (состоят из N объектов; при описании класса, которому принадлежит такой объект, указывается только одна именованная переменная, остальные N-1 индексируются и доступ к ним осуществляется посылкой сообщения "вПозиции: индекс", а не указанием имен).

Объекты с индексируемыми переменными также могут иметь поименованные переменные экземпляра.

Набор - это группа связанных между собой объектов. Классы наборов определяют различные структуры данных для хранения произвольных объектов. Основные подклассы класса «Набор» представлены на рис.3.19.

Полиморфизм проявляется в том, что одно и то же сообщение может посылаться разным объектам, и понимают они его каждый по-своему. Это увеличивает наглядность программ, поскольку не надо беспокоиться о случайном совпадении имен сообщений, как это имеет место с именами функций в традиционных языках.

Классы также являются объектами. Таким образом, классам тоже можно посылать сообщения. Поскольку сообщение, посылаемое экземпляру, анализируется этим классом, то логично предположить, что сообщение, посылаемое классу, анализировалось его классом. Мы получили единый механизм вызова сообщений, обеспечивающий ясность программ.

Класс класса называется **метаклассом**. Классам присваиваются имена, а метаклассам - нет. Для каждого класса существует один метакласс (рис. 3.20).

Между метаклассами тоже существует иерархия, образованная тем же отношением «супер-класс - подкласс». Иерархия классов и иерархия метаклассов изоморфны друг другу.

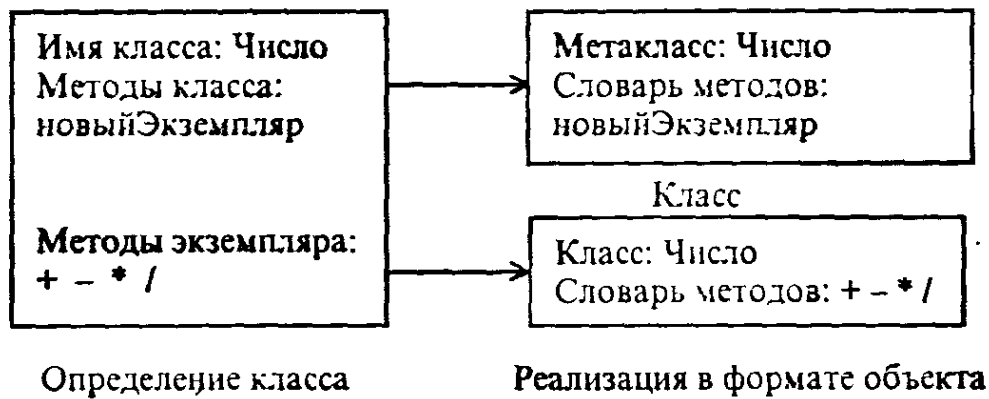


Рис. 3.20. Классы и метаклассы

В ряде случаев необходимо работать с объектами, обладающими свойствами пересечения некоторых классов, но не включения, т.е.:

Класс А не является подклассом класса В

Класс В не является подклассом класса А Пересечение А и В не пусто.

В таком случае создается специальный суперкласс обоих классов, описывающий их общую часть. Этот суперкласс называют абстрактным классом. Он не создает своих экземпляров, а только служит для наследования общих методов.

Рекурсии. Как и в других языках, ориентированных на обработку символов и нашедших применение в создании систем искусственного интеллекта, в SmallTalk важную роль играет рекурсия. Рассмотрим пример определения рекурсивного метода, вычисляющего числа Фибоначчи:

Фибоначчи

"выдать N чисел Фибоначчи, где N - объект-адресат"

сам<3

еслиИстина: [^ 1]

еслиЛожь : [^ (сам - 1) фибоначчи + (сам - 2) фибоначчи]

Классическим примером задачи, требующей рекурсии, является задача о Ханойских башнях. Имеется три штырька. На штырьке №1 расположены кольца разного диаметра в порядке убывания их размера снизу вверх. Требуется переложить кольца со штырька №1 на штырек №2, пользуясь штырьком №3 как промежуточным, таким образом, чтобы кольца на штырьке №2 располагались в порядке убывания их диаметра снизу вверх и чтобы в процессе перекладывания кольцо большего размера никогда не накладывалось на кольцо меньшего диаметра.

Данная задача решается рекурсивно.

а) Базис.

Если кольцо одно, то надо просто переложить это кольцо с x на y.

б) Рекурсия.

Если число колец равно k, то надо переложить со штырька x на штырек z k-1 колец, затем переложить самое большое k-е кольцо на y и переложить все остальные k-1 колец со штырька z на y.

Объект-адресат в нашем случае - целое число (количество колец). (Следовательно, данный метод будет принадлежать классу целых чисел.)

Объекты-параметры - символы '1', '2', '3' (информация: с какого штырька на какой надо переложить кольцо, а какой использовать как дополнительный).

Результат - надписи на экране о последовательности выполнения данной задачи.

Следовательно, вызов метода будет осуществляться следующим образом:

<число> ханойС: '1' на: '2' через: '3'

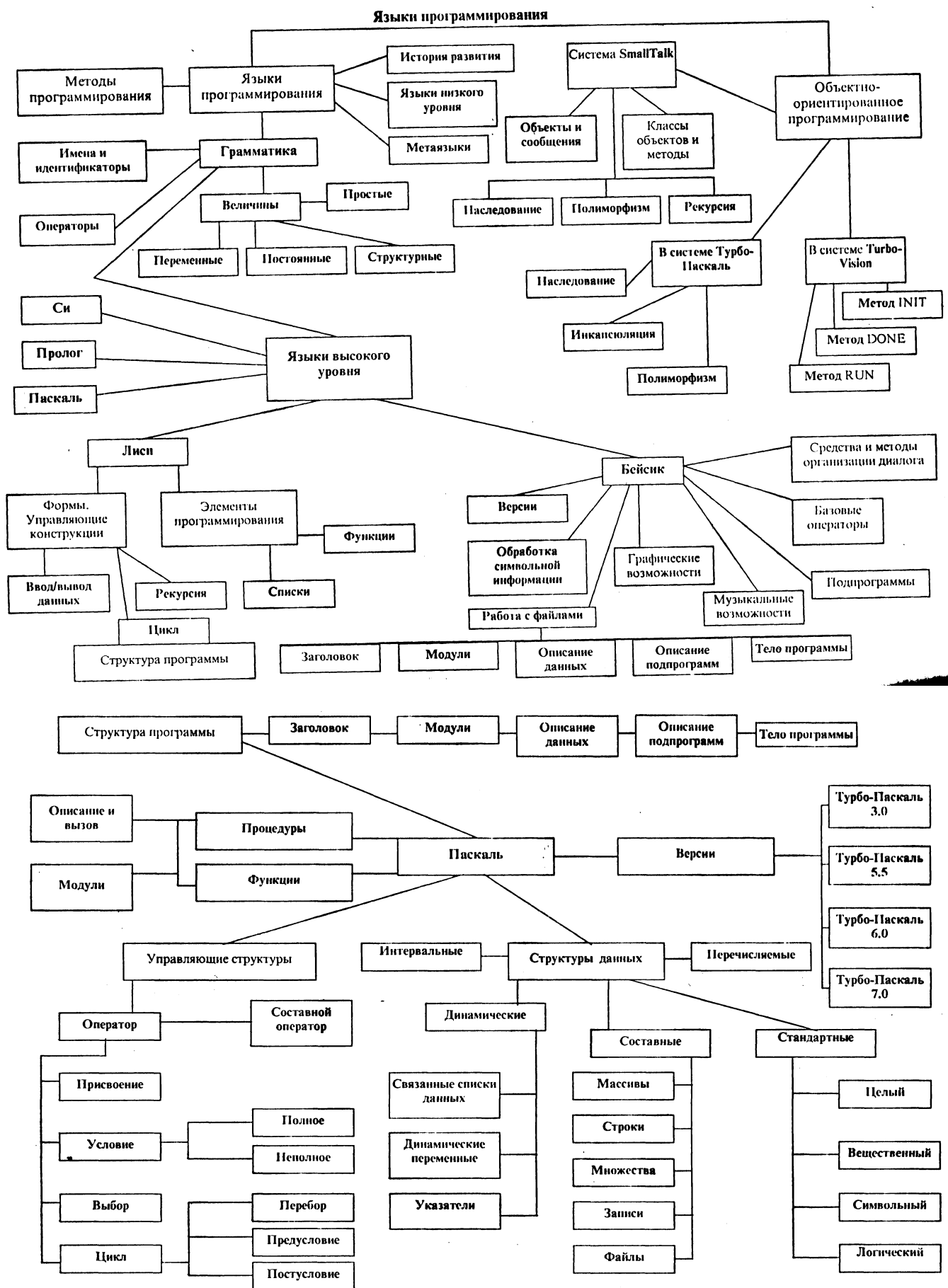
Реализация метода ханойС: x на: y через: z может быть следующей:

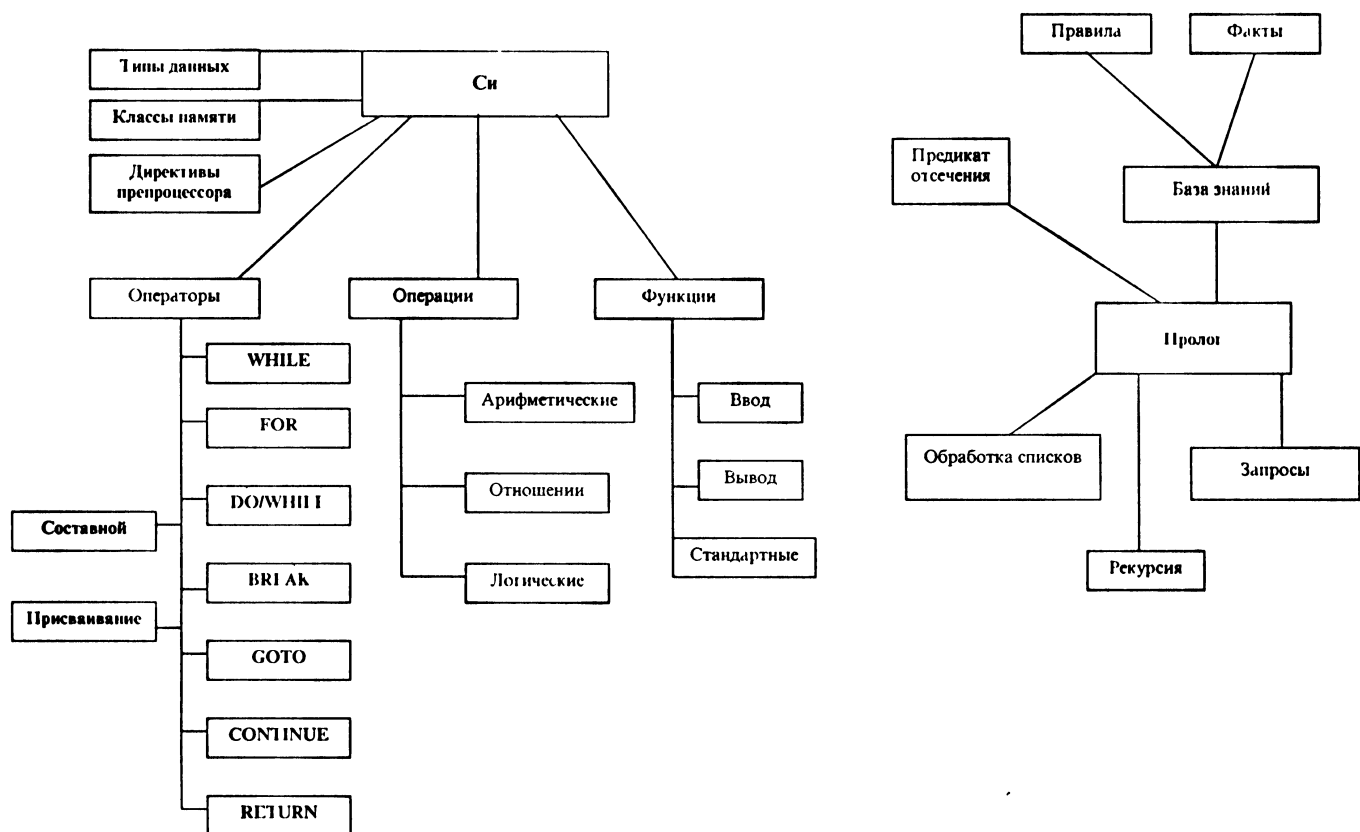
ханойС: х на: у через: z
"головоломка 'Ханойские башни'"
сам=1

еслиИстина: [СистемнаяИнформация поместитьВсеПоследующие:
'Переложить со штырька',х,'на штырек',у;символВК.]
еслиЛожь: [(сам - 1) ханойС: х на: z через: у
СистемнаяИнформация поместитьВсеПоследующие:
'Переложить со штырька',х,'на штырек',у;символВК.]
(сам - 1) ханойС: z на: у через: х]

Контрольные вопросы и задания

1. Какие методологии могут быть использованы при проектировании программных систем?
2. В чем состоит смысл объектно-ориентированной методологии проектирования программ?
3. Каковы основные шаги разработки программы в объектно-ориентированной методологии?
4. Как описываются объекты в Турбо-Паскале?
5. В чем отличие методов объектов от обычных процедур? Как методы задаются?
6. Что такое инкапсуляция, наследование и полиморфизм? Приведите примеры.
7. Каково назначение и возможности объектно-ориентированной оболочки Turbo-Vision?
8. Охарактеризуйте основные объекты и методы Turbo-Vision.
9. В чем состоят отличия визуального объектного программирования от более традиционного?
10. Каковы концепции, положенные в основу языка SmallTalk?
11. Какие типы сообщения возможны в языке SmallTalk?
12. Охарактеризуйте основные встроенные в SmallTalk классы.
13. Какие управляющие конструкции существуют в языке SmallTalk?
14. Как определяются новые методы? объекты?
15. Как организуется рекурсия в программах на SmallTalk?





Дополнительная литература к главе 3

1. *Абрамов С. А. и др. Задачи по программированию.* - М.: Наука, 1988.
2. *Абрамов В. Г., Трифонов Н.П., Трифонова Г.Н. Введение в язык Паскаль.* - М.: Наука, 1988.
3. *Алексеев В.Е. и др. Вычислительная техника и программирование. Практикум по программированию.* - М.: Высшая школа, 1991.
4. *Братка И. Программирование на языке Пролог для искусственного интеллекта.* - М.: Мир, 1990.
5. *Брукс Ф.П. Как проектируются и создаются программные комплексы. /Очерки по системному программированию/.* - М.: Наука, 1979.
6. *Буч Г. Объектно-ориентированное программирование с примерами применения.* - Киев: Диалектика, 1992.
7. *Ван Тассел Д. Стил, разработка, эффективность, отладка и испытание программ.* - М.: Мир, 1981.
8. *Вирт Н. Алгоритмы и структуры данных.* - М.: Мир, 1989.
9. *Вирт Н. Алгоритмы + структура данных = программы.* - М.: Мир, 1985.
10. *Герман О. В. Введение в теорию экспертных систем и обработку знаний.* - Минск, Дизайн-ПРО, 1995.
11. *Гудман С, Хидетниemi С. Введение в разработку и анализ алгоритмов.* - М.: Мир, 1981.
12. *Дантемманн Д., Мишел Д., Тейлор Д. Программирование в среде Delphy.* - Киев: НИПФ-ДиаСофтЛтд., 1995.
13. *Дарахавелидзе П., Марков Е. Delphy - среда визуального программирования.* - СПб.: BVH-Санкт-Петербург, 1996.
14. *Дайтибегов Д.М., Черноусое Е.А. Основы алгоритмизации и алгоритмические языки.* - М.: Финансы и статистика, 1992.
15. *Джонс Ж., Харроу К. Решение задач в системе Турбо-Паскаль.* - М.: Финансы и статистика, 1991.
16. *Дмитриева М.В., Кубенский А.А. Элементы современного программирования.* - С.-Пб.: Изд-во С.-П. университета, 1991.

17. Довгаль С.И., Литейное Б.Ю., Сбитнев А. Персональные ЭВМ: Турбо-Паскаль V 7.0. Объектное программирование. Локальные сети./Уч. пособие. - Киев: Информсистема сервис, 1993.
18. Зуев Е.А. Программирование на языке TURBO PASCAL 6.0 - 7.0. - М.: Радио и связь, 1993.
19. Зуев Е.А. Практическое программирование на языке Турбо-Паскаль о.и, 7.0. -М.: Радио и связь, 1994.
20. Кетков Ю.Л. Диалог на языке бейсик для мини- и микро-ЭВМ. - М.: Наука, 1988.
21. Кнут Д. Искусство программирования. Т. 1, 2, 3. - М.: Мир, 1976 - 1978.
22. Липаев В. В. Проектирование программных средств. - М.: Высшая школа, 1990.
23. Ляхович В. Ф. Руководство к решению задач по основам информатики и вы-- делительной техники. - М.: Высшая школа, 1994.
24. Майерс Г. Искусство тестирования программ. - М.: Финансы и статистика, 1982.
25. МалпасДж. Реляционный язык ПРОЛОГ и его применение. - М.: Наука, 1990.
26. Пильщиков В. Н. Сборник упражнений по языку Паскаль. - М.: Наука, 1989.
27. Поляков Д. Б., Круглое И. Ю. Программирование в среде Турбо-Паскаль. - М.: А/О Росвузнаука, 1992.
28. Роджерс Д. Ф., Адамс Дж.А. Математические основы машинной графики. -М: Машиностроение, 1980.
29. Рубенкинг И. Турбо-Паскаль для Windows. Т.1,2. - М.: Мир-СК Ферлаг Ин-тернешнл, 1994.
30. Семакин И. Г. Лекции по программированию. - Пермь: Изд-во ПГУ, 1996.
31. Сергиевский М. В., Шалашов А. В. Турбо-Паскаль 7.0. - М.: Машиностроение, 1994.
32. Уэйт М., Прата С., Мартин Д. Язык Си. - М.: Мир, 1988.
33. Федоров А., РогаткинД. Borland Pascal в среде Windows. - Киев: Диалектика, 1993.
34. ФоксДж. Программное обеспечение и его разработка. - М.: Мир, 1985.
35. Хендерсон. Функциональное программирование.
36. Хьюз Дж., Мичтом Дж. Структурный подход к программированию. - М.: Мир, 1980.
37. Шикин Е. В. Начала компьютерной графики. - М.: Диалог-МИФИ, 1994.
38. Язык компьютера. SoftWare. Computer languages./Пер. с англ. Под ред. В.М.Курочкина. - М.: Мир, 1989.

ЧАСТЬ ВТОРАЯ

ГЛАВА 4

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

«... Ибо это недостойно совершенства Человеческого - подобно рабам тратить часы на вычисления».

Готфрид Вильгельм Лейбниц

ВВЕДЕНИЕ

Настоящая глава посвящена принципиальным вопросам устройства и функционирования ЭВМ.

В познании деятельности компьютера есть несколько уровней. Первый из них, необходимый каждому специалисту, - уровень архитектуры. Архитектура - это наиболее общие принципы построения ЭВМ, реализующие программное управление работой и взаимодействием основных функциональных узлов. На этом уровне не требуется знание схемных решений современной радиотехники и микроэлектроники. Последнее вообще выходит за пределы информатики, оно требуется лишь разработчикам физических элементов компьютеров.

Уровень архитектуры достаточно глубок, он включает вопросы управления работой ЭВМ (программирования) на языке машинных команд (ассемблера). Такой способ управления гораздо сложнее, чем написание программ на языках высокого уровня, и тем не менее без представления о нем невозможно понять реальную работу компьютера.

Следующий уровень, который также прослеживается в данной главе, - логические принципы и схемы реализации основных операционных узлов компьютера (триггеров, сумматоров и т.д.). Понимание этих принципов весьма желательно и существенно расширит кругозор специалиста в области информатики (и ее преподавания).

Наконец, в данной главе приведен краткий обзор внешних устройств современных компьютеров - накопителей, устройств ввода и вывода информации, - а также элементарное описание принципов их работы, профессиональные характеристики.

§ 1. ИСТОРИЯ РАЗВИТИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

1.1. НАЧАЛЬНЫЙ ЭТАП РАЗВИТИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Все началось с идеи научить машину считать или хотя бы складывать многозначные целые числа. Еще около 1500 г. великий деятель эпохи Просвещения Леонардо да Винчи разработал эскиз 13-разрядного суммирующего устройства, что явилось первой дошедшей до нас попыткой решить указанную задачу. Первую же действующую суммирующую машину построил в 1642 г. Блез Паскаль - знаменитый французский физик, математик, инженер. Его 8-разрядная машина сохранилась до наших дней.

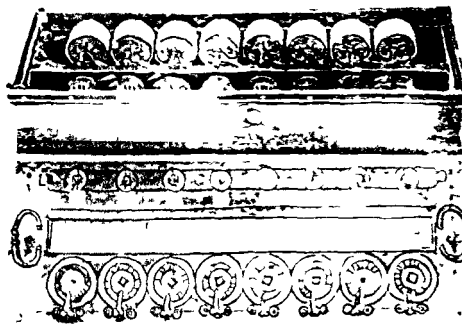


Рис. 4.1. Блез Паскаль (1623 - 1662) и его счетная машина

От замечательного курьеза, каким восприняли современники машину Паскаля, до создания практически полезного и широко используемого агрегата - арифмометра (механического вычислительного устройства, способного выполнять 4 арифметических действия) - прошло почти 250 лет. Уже в начале XIX века уровень развития ряда наук и областей практической деятельности (математики, механики, астрономии, инженерных наук, навигации и др.) был столь высок, что они настоятельнейшим образом требовали выполнения огромного объема вычислений, выходящих за пределы возможностей человека, не вооруженного соответствующей техникой. Над ее созданием и совершенствованием работали как выдающиеся ученые с мировой известностью, так и сотни людей, имена многих из которых до нас не дошли, посвятивших свою жизнь конструированию механических вычислительных устройств.

Еще в 70-х годах нашего века на полках магазинов стояли механические арифмометры и их «ближайшие родственники», снабженные электрическим приводом - электромеханические клавишные вычислительные машины. Как это часто бывает, они довольно долго удивительным образом соседствовали с техникой совершенно иного уровня - автоматическими цифровыми вычислительными машинами (АЦВМ), которые в просторечии чаще называют ЭВМ (хотя, строго говоря, эти понятия не совсем совпадают). История АЦВМ восходит еще к первой половине прошлого века и связана с именем замечательного английского математика и инженера Чарльза Бэббиджа. Им в 1822 г. была спроектирована и почти 30 лет строилась и совершенствовалась машина, названная вначале «разностной», а затем, после многочисленных усовершенствований проекта, «аналитической». В «аналитическую» машину были заложены принципы, ставшие фундаментальными для вычислительной техники.

1. Автоматическое выполнение операций.

Для выполнения расчетов большого объема существенно не только то, как-быстро выполняется отдельная арифметическая операция, но и то, чтобы между операциями не было «зазоров», требующих непосредственного человеческого вмешательства. Например, большинство современных калькуляторов не удовлетворяют этому требованию, хотя каждое доступное им действие выполняют очень быстро. Необходимо, чтобы операции следовали одна за другой безостановочно.

2. Работа по вводимой «на ходу» программе.

Для автоматического выполнения операций программа должна вводиться в исполнительное устройство со скоростью, соизмеримой со скоростью выполнения операций. Бэббидж предложил использовать для предварительной записи программ и ввода их в машину перфокарты, которые к тому времени применялись для управления ткацкими станками.

3. Необходимость специального устройства - памяти - для хранения данных (Бэббидж назвал его «складом»).

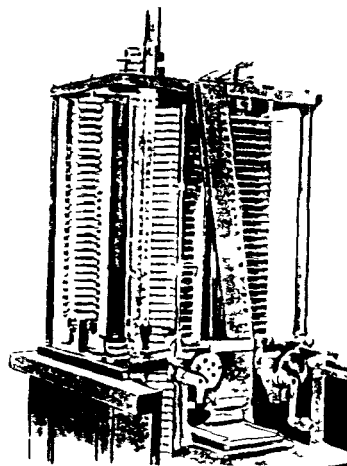
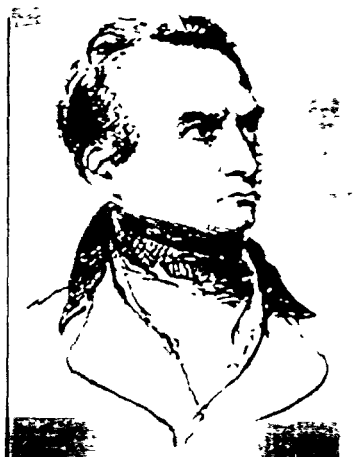


Рис. 4.2. Чарльз Бэббидж (1792 - 1871) и его «аналитическая машина»

Эти революционные идеи натолкнулись на невозможность их реализации на основе механической техники, ведь до появления первого электромотора оставалось почти полвека, а первой электронной радиолампы - почти век! Они настолько опередили свое время, что были в значительной мере забыты и переоткрыты в следующем столетии.

Впервые автоматически действующие вычислительные устройства появились в середине XX века. Это стало возможным благодаря использованию наряду с механическими конструкциями электромеханических реле. Работы над релейными машинами начались в 30-е годы и продолжались с переменным успехом до тех пор, пока в 1944 г. под руководством Говарда Айкена - американского математика и физика - на фирме IBM (International Business Machines) не была запущена машина «Марк-1», впервые реализовавшая идеи Бэббиджа (хотя разработчики, по-видимому, не были с ними знакомы). Для представления чисел в ней были использованы механические элементы (счетные колеса), для управления - электромеханические. Одна из самых мощных релейных машин РВМ-1 была в начале 50-х годов построена в СССР под руководством Н.И.Бессонова; она выполняла до 20 умножений в секунду с достаточно длинными двоичными числами.

Однако, появление релейных машин безнадежно запоздало и они были очень быстро вытеснены электронными, гораздо более производительными и надежными.

1.2. НАЧАЛО СОВРЕМЕННОЙ ИСТОРИИ ЭЛЕКТРОННОЙ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Подлинная революция в вычислительной технике произошла в связи с применением электронных устройств. Работа над ними началась в конце 30-х годов одновременно в США, Германии, Великобритании и СССР. К этому времени электронные лампы, ставшие технической основой устройств обработки и хранения цифровой информации, уже широчайшим образом применялись в радиотехнических устройствах.

Первой действующей ЭВМ стал ENIAC (США, 1945 - 1946 гг.). Его название по первым буквам соответствующих английских слов означает «электронно-числовой интегратор и вычислитель». Руководили ее созданием Джон Моучли и Преспер Эккерт, продолжившие начатую в конце 30-х годов работу Джорджа Атанасова. Машина содержала порядка 18 тысяч электронных ламп, множество электромеханических элементов. Ее энергопотребление равнялось 150 кВт, что вполне достаточно для обеспечения небольшого завода.

Практически одновременно велись работы над созданием ЭВМ в Великобритании. С ними связано прежде всего имя Аллана Тьюринга - математика, внесшего также большой вклад в теорию алгоритмов и теорию кодирования. В 1944 г. в Великобритании была запущена машина «Колосс».

Эти и ряд других первых ЭВМ не имели важнейшего с точки зрения конструкторов последующих компьютеров качества - программа не хранилась в памяти машины, а набиралась достаточно сложным образом с помощью внешних коммутирующих устройств.

Огромный вклад в теорию и практику создания электронной вычислительной техники на

начальном этапе ее развития внес один из крупнейших американских математиков Джон фон Нейман. В историю науки навсегда вошли «принципы фон Неймана». Совокупность этих принципов породила классическую (фон-неймановскую) архитектуру ЭВМ. Один из важнейших принципов - принцип хранимой программы - требует, чтобы программа закладывалась в память машины так же, как в нее закладывается исходная информация. Первая ЭВМ с хранимой программой (EDSAC) была построена в Великобритании в 1949 г.



Рис. 4.3. Джон фон Нейман (1903-1957)



Рис. 4.4. Сергей Александрович Лебедев (1902- 1974)

В нашей стране вплоть до 70-х годов создание ЭВМ велось почти полностью самостоятельно и независимо от внешнего мира (да и сам этот «мир» был почти полностью зависим от США). Дело в том, что электронная вычислительная техника с самого момента своего первоначального создания рассматривалась как сверхсекретный стратегический продукт, и СССР приходилось разрабатывать и производить ее самостоятельно. Постепенно режим секретности смягчался, но и в конце 80-х годов наша страна могла покупать за рубежом лишь устаревшие модели ЭВМ (а самые современные и мощные компьютеры ведущие производители - США и Япония - и сегодня разрабатывают и производят в режиме секретности).

Первая отечественная ЭВМ - МЭСМ («малая электронно-счетная машина») - была создана в 1951 г. под руководством Сергея Александровича Лебедева, крупнейшего советского конструктора вычислительной техники, впоследствии академика, лауреата государственных премий, руководившего созданием многих отечественных ЭВМ. Рекордной среди них и одной из лучших в мире для своего времени была БЭСМ-6 («большая электронно-счетная машина, 6-я модель»), созданная в середине 60-х годов и долгое время бывшая базовой машиной в обороне, космических исследованиях, научно-технических исследованиях в СССР. Кроме машин серии БЭСМ выпускались и ЭВМ других серий - «Минск», «Урал», М-20, «Мир» и другие, созданные под руководством И.С.Брука и М.А.Карцева, Б.И.Рамеева, В.М.Глушкова, Ю.А.Базилевского и других отечественных конструкторов и теоретиков информатики.

С началом серийного выпуска ЭВМ начали условно делить по поколениям; соответствующая классификация изложена ниже.

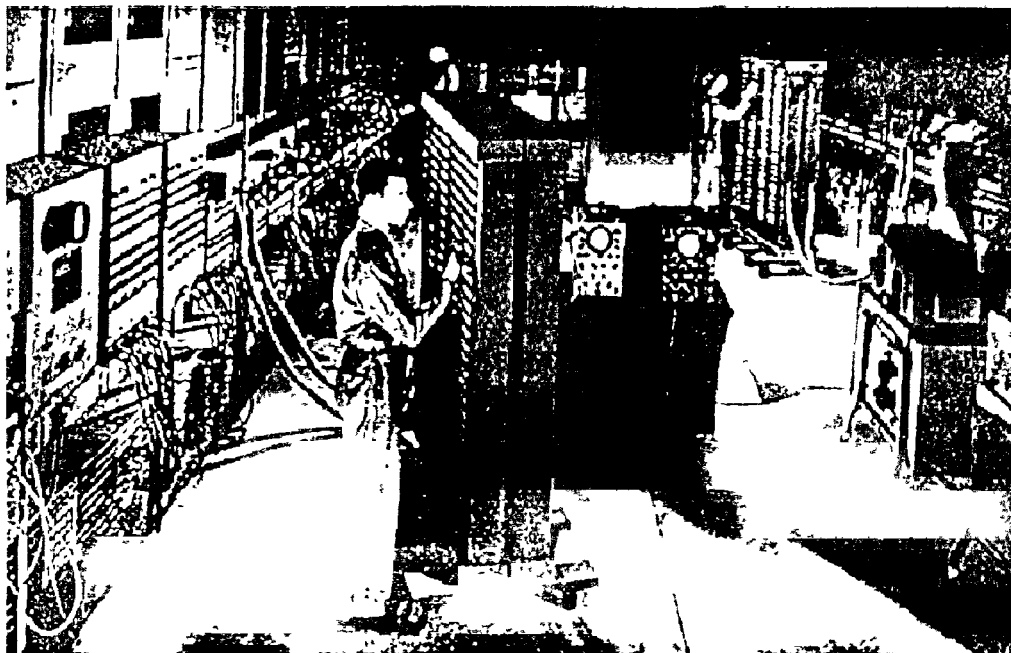


Рис. 4.5. Первая в мире ЭВМ ENIAC

1.3. ПОКОЛЕНИЯ ЭВМ

В истории вычислительной техники существует своеобразная периодизация ЭВМ по поколениям. В ее основу первоначально был положен физико-технологический принцип: машину относят к тому или иному поколению в зависимости от используемых в ней физических элементов или технологии их изготовления. Границы поколений во времени размыты, так как в одно и то же время выпускались машины совершенно разного уровня. Когда приводят даты, относящиеся к поколениям, то скорее всего имеют в виду период промышленного производства; проектирование велось существенно раньше, а встретить в эксплуатации весьма экзотические устройства можно и сегодня.

В настоящее время физико-технологический принцип не является единственным при определении принадлежности той или иной ЭВМ к поколению. Следует считаться и с уровнем программного обеспечения, с быстродействием, другими факторами, основные из которых сведены в прилагаемую табл. 4.1.

Следует понимать, что разделение ЭВМ по поколениям весьма относительно. Первые ЭВМ, выпускавшиеся до начала 50-х годов, были «штучными» изделиями, на которых отрабатывались основные принципы; нет особых оснований относить их к какому-либо поколению. Нет единодушия и при определении признаков пятого поколения. В середине 80-х годов считалось, что основной признак этого (будущего) поколения - полновесная реализация принципов искусственного интеллекта. Эта задача оказалась значительно сложнее, чем виделось в то время, и ряд специалистов снижают планку требований к этому этапу (и даже утверждают, что он уже состоялся). В истории науки есть аналоги этого явления: так, после успешного запуска первых атомных электростанций в середине 50-х годов ученые объявили, что запуск многократно более мощных, дающих дешевую энергию, экологически безопасных термоядерных станций, вот-вот произойдет; однако, они недооценили гигантские трудности на этом пути, так как термоядерных электростанций нет и по сей день.

В то же время среди машин четвертого поколения разница чрезвычайно велика, и поэтому в табл. 4.1 соответствующая колонка разделена на две: А и Б. Указанные в верхней строчке даты соответствуют первым годам выпуска ЭВМ. Многие понятия, отраженные в таблице, будут обсуждаться в последующих разделах учебника; здесь ограничимся кратким комментарием.

Чем младше поколение, тем отчетливее классификационные признаки. ЭВМ первого, второго и третьего поколений сегодня, в конце 90-х годов - в лучшем случае музейные экспонаты. Машина первого поколения - десятки стоек, каждая размером с большой книжный шкаф, наполненных электронными лампами, лентопротяжными устройствами, громоздкие печатающие агрега-

ты, и все это на площади сотни квадратных метров, со специальными системами охлаждения, источниками питания, постоянно гудящее и вибрирующее (почти как в цехе машиностроительного завода). Обслуживание - ежечасное. Часто выходящие из строя узлы, перегорающие лампы, и вместе с тем невиданные, волшебные возможности для тех, кто, например, занят математическим моделированием. Быстродействие до 1000 оп/с и память на 1000 чисел делало доступным решение задач, к которым раньше нельзя было и подступиться.

Приход полупроводниковой техники (первый транзистор был создан в 1948 г., а первая ЭВМ с их использованием - в 1956 г.) резко изменил вид машинного зала -более нормальный температурный режим, меньший гул (лишь от внешних устройств) и, самое главное, возросшие возможности для пользователя. Впрочем, непосредственного пользователя к машинам первых трех поколений почти никогда

Таблица 4.
Поколения ЭВМ

Показатель	Поколения ЭВМ					
	Первое	Второе	Третье	Четвертое		Пятое
	1951-1954	1958-1960	1965-1966	А 1976-1979	Б 1985-?	?
Элементная база процессора	Электронные лампы	Транзисторы	Интегральные схемы (ИС)	Большие ИС (БИС)	Сверхбольшие ИС (СБИС)	+Оптоэлектроника +Криоэлектроника
Элементная база ОЗУ	Электронно-лучевые трубки	Ферритовые сердечники	Ферритовые сердечники	БИС	СБИС	СБИС
Максимальная емкость ОЗУ, байт	10^2	10^1	10^4	10^5	10^7	10^8 (?)
Максимальное быстродействие процессора (оп/с)	10^4	10^6	10^7	10^8	10^9 +Многопроцессорность	10^{12} , +Многопроцессорность
Языки программирования	Машинный код	+ Ассемблер	+ Процедурные языки высокого уровня (ЯВУ)	+ Новые процедурные ЯВУ	+Непроцедурные ЯВУ	+ Новые непрцедурные ЯВУ
Средства связи пользователя с ЭВМ	Пульт управления и перфокарты	Перфокарты и перфоленты	Алфавитно-цифровой терминал	Монохромный графический дисплей, клавиатура	Цветной + графический дисплей, клавиатура, «мышь» и др.	Устройства голосовой связи с ЭВМ

не подпускали - около них колдовали инженеры, системные программисты и операторы, а пользователь чаще всего передавал в узкое окошечко или клал на стеллаж в соседнем помещении рулон перфоленты или колоду перфокарт, на которых была его программа и входные данные задачи. Доминировал для машин первого и второго поколения монопольный режим пользования машиной и/или режим пакетной обработки; в третьем поколении добавился более выгодный экономически и более удобный для пользователей удаленный доступ - работа через выносные терминалы в режиме разделения времени.

Уже начиная со второго поколения, машины стали делиться на большие, средние и малые

по признакам размеров, стоимости, вычислительных возможностей. Так, небольшие отечественные машины второго поколения («Наири», «Раздан», «Мир» и др.) с производительностью порядка 10^4 оп/с были в конце 60-х годов вполне доступны каждому вузу, в то время как упомянутая выше БЭСМ-6 имела профессиональные показатели (и стоимость) на 2 - 3 порядка выше.

В начале 70-х годов, с появлением интегральных технологий в электронике, были созданы микроэлектронные устройства, содержащие несколько десятков транзисторов и резисторов на одной небольшой (площадью порядка 1 см^2) кремниевой подложке. Без пайки и других привычных тогда в радиотехнике действий на них «выращивались» электронные схемы, выполняющие функции основных логических узлов ЭВМ (триггеры, сумматоры, дешифраторы, счетчики и т.д.). Это позволило перейти к третьему поколению ЭВМ. технической база которого - интегральные схемы.

При продвижении от первого к третьему поколению радикально изменились возможности программирования. Написание программ в машинном коде для машин первого поколения (и чуть более простое на Ассемблере) для большей части машин второго поколения является занятием, с которым подавляющее большинство современных программистов знакомятся при обучении в вузе, а потом забывают. Появление процедурных языков высокого уровня и трансляторов с них было первым шагом на пути радикального расширения круга программистов. Научные работники и инженеры сами стали писать программы для решения своих задач.

Уже в третьем поколении появились крупные унифицированные серии ЭВМ. Для больших и средних машин в США это прежде всего семейство IBM 360/370. В СССР 70-е и 80-е годы были временем создания унифицированных серии: ЕС (единая система) ЭВМ (крупные и средние машины), СМ (система малых) ЭВМ и «Электроника» (серия микро-ЭВМ). В их основу были положены американские прототипы фирм IBM и DEC (Digital Equipment Corporation). Были созданы и выпущены десятки моделей ЭВМ, различающиеся назначением и производительностью. Их выпуск был практически прекращен в начале 90-х годов, но многие из них еще используются в самых разных сферах деятельности, включая образование (например, компьютеры ДВК, БК, а также УКНЦ - аналоги мини-ЭВМ типа PDP-11 фирмы DEC).

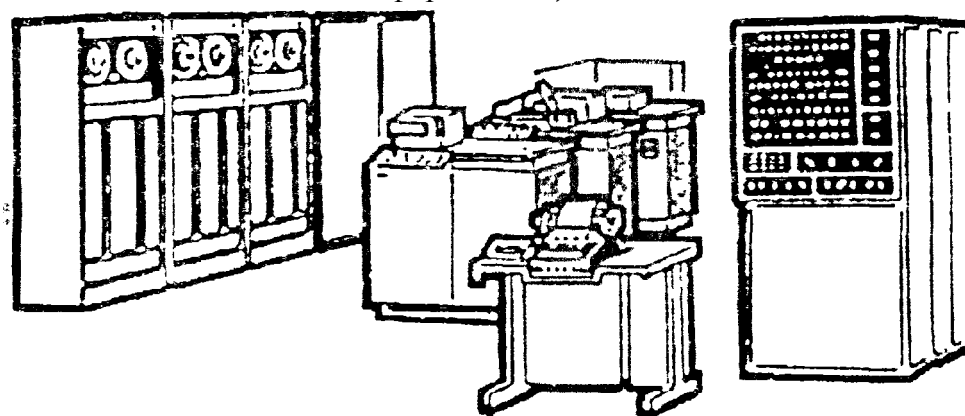


Рис. 4.6. ЭВМ третьего поколения

1.4. ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ

Подлинную революцию в вычислительной технике произвело создание микропроцессора. В 1971 г. компанией «Intel» (США) было создано устройство, реализующее на одной крошечной микросхеме функции процессора - центрального узла ЭВМ. Последствия этого оказались огромны не только для вычислительной техники, но и для научно-технического прогресса в целом. В области разработки ЭВМ первым таким последствием оказалось создание персональных компьютеров (ПК) - небольших и относительно недорогих ЭВМ, способных аккумулировать и усиливать интеллект своего персонального хозяина (впрочем, заметим, что как и всякое техническое средство, ПК способен и на обратный эффект - напрасно отнимать время и подавлять интеллект).

Небольшие компьютеры, предназначенные для одного пользователя, который в каждый момент решает не более одной задачи, использовались в профессиональной деятельности уже в начале 70-х годов. Восьмиразрядные микропроцессоры i8080 и Z80 в сочетании с операционной системой CP/M позволили создать ряд таких компьютеров, но тем не менее началом эры их массового появления стал 1976 г., когда появился знаменитый «Apple» («Яблоко»), созданный молоды-

ми американскими инженерами Стивом Возняком и Стивом Джобсом. За несколько лет было продано около 2 млн. экземпляров лишь этих ПК (особенно «Apple-2»), т.е. впервые в мировой практике компьютер стал устройством массового производства. Вскоре лидерство в этой области захватила фирма IBM - компьютерный гигант, представивший в 1981 г. свой персональный компьютер IBM PC (PC - persona computer). Его модели PC XT (1983 г.), PC AT (1984 г.), ПК с микропроцессором Pentium (начало 90-х годов; содержит более 3 миллионов транзисторов!) стали, каждый в свое время, ведущими на мировом рынке ПК. В настоящее время производство ПК ведут десятки фирм (а комплектующие выпускают сотни фирм) по всему миру.

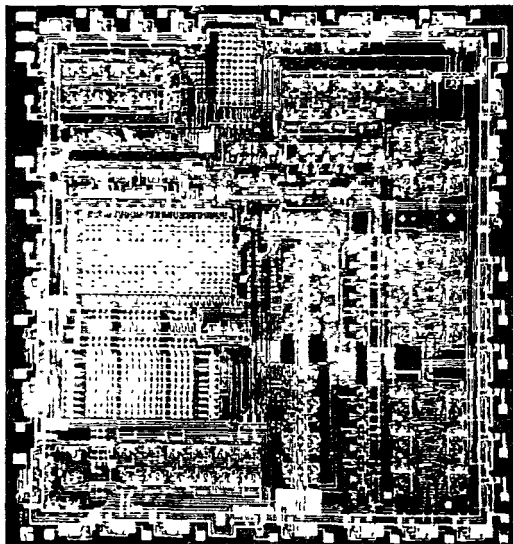


Рис. 4.7. Микропроцессор (сильно увеличенная фотография в разрезе)

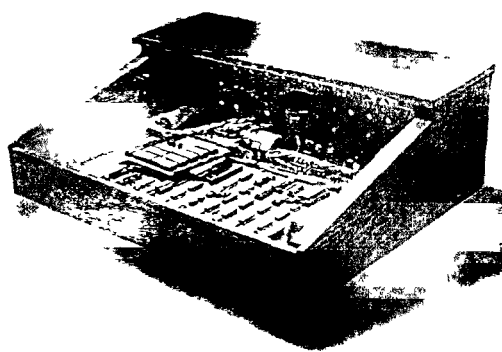


Рис. 4.8. Первый персональный компьютер «Apple»

Ближайшим конкурентом компьютеров IBM PC являются персональные компьютеры фирмы «Apple Computer». Пришедшие на смену «Apple-2» машины «Macintosh» широко используются в системах образования многих стран.

В дальнейшем, по мере знакомства с архитектурой ЭВМ, рассказ о ПК будет продолжен. Сейчас же уточним характеристики, которые в совокупности позволяют отнести компьютер к этой группе:

- относительно невысокая стоимость (доступная для приобретения в личное пользование значительной частью населения);
- наличие «дружественных» операционной и интерфейсной систем, которые максимально упрощают пользователю работу с компьютером;
- наличие достаточно развитого и относительно недорогого набора внешних устройств в «настольном» исполнении;
- наличие аппаратных и программных ресурсов общего назначения, позволяющих решать реальные задачи по многим видам профессиональной деятельности.

За четверть века, прошедшие с момента создания ПК, уже сменилось несколько их поколений: 8-битные, 16-битные, 32-битные. Многократно усовершенствовались внешние устройства, все операционное окружение, включая сети, системы связи, системы программирования, программное обеспечение и т.д. Персональный компьютер занял нишу «персонального усилителя интеллекта» множества людей, стал в ряде случаев ядром автоматизированного рабочего места (в цехе, в банке, в билетной кассе, в школьном классе - все перечислить невозможно).

1.5. И НЕ ТОЛЬКО ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ...

Массовость использования ПК, огромные рекламные усилия производителей и коммерсантов не должны заслонить тот факт, что кроме ПК есть и другие, многократно более мощные, вычислительные системы. Всегда есть круг задач, для которых недостаточно существующих вычислительных мощностей и которые столь важны, что для их решения не жалко никаких средств. Это,

например, может быть связано с обороноспособностью государства, решением сложнейших научно-технических задач, созданием и поддержкой гигантских банков данных. В настоящее время лишь немногие государства способны производить, так называемые, супер-ЭВМ - компьютеры, на фоне которых «персоналки» кажутся игрушками. Впрочем, сегодня ПК часто становится терминалом - конечным звеном в гигантских телекоммуникационных системах, в которых решением непосильных для ПК задач обработки информации занимаются более мощные ЭВМ.

Схема классификации компьютеров, исходящая из их производительности, размеров и функционального назначения, приведена на рис. 4.9. Следует отметить, что вопрос об отнесении конкретного компьютера к одной из категорий этой схемы может иметь неоднозначный ответ, привязанный к конкретной исторической обстановке или доминирующему поколению ЭВМ.

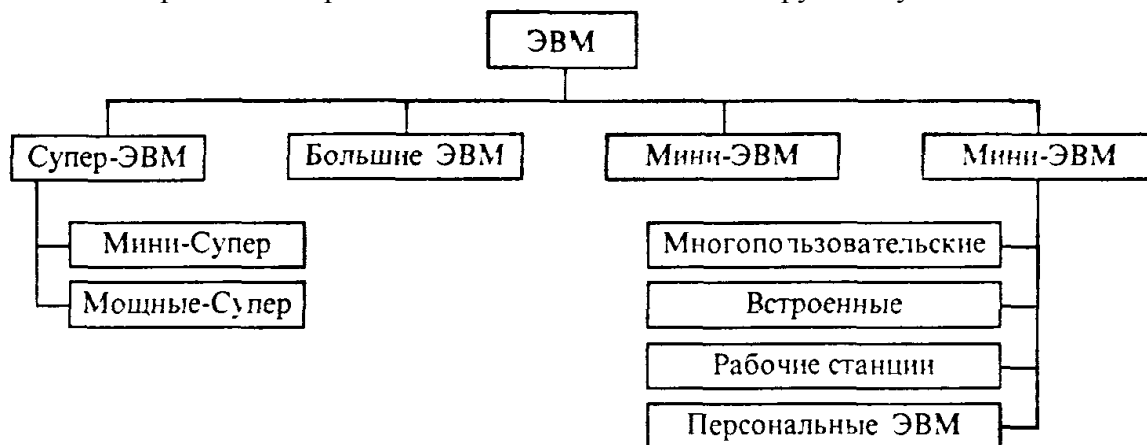


Рис. 4.9. Классификация ЭВМ

Место супер-ЭВМ в этой иерархии уже обсуждалось. Определить супер-ЭВМ можно лишь относительно: это самая мощная вычислительная система, существующая в соответствующий исторический период. В настоящее время наиболее известны мощные супер-ЭВМ «Cray» и «IBM SP2» (США). Модель «Cray-3», выпускаемая с начала 90-х годов на основе принципиально новых микроэлектронных технологий, является 16-процессорной машиной с быстродействием более 10 млрд. операций в секунду (по другим данным 16) над числами с «плавающей точкой» (т.е. длинными десятичными числами; такие операции гораздо более трудоемки, чем над целыми числами); в модели CS 6400 число процессоров доведено до 64. Супер-ЭВМ требуют особого температурного режима, зачастую водяного охлаждения (или даже охлаждения жидким азотом). Их производство по масштабам несопоставимо с производством компьютеров других классов (так, в 1995 г. корпорацией «Cray» было выпущено всего около 70 таких компьютеров).

Большие ЭВМ более доступны, чем «супер». Они также требуют специального помещения, иногда весьма немалого, поддержания жесткого температурного режима, высококвалифицированного обслуживания. Такую ЭВМ в 80-е годы мог себе позволить завод, даже крупный вуз. Классическим примером служат выпускавшиеся еще недавно в США машины серии IBM 370 и их отечественные аналоги ЕС ЭВМ. Большие ЭВМ используются для производства сложных научно-технических расчетов, математического моделирования, а также в качестве центральных машин в крупных автоматизированных системах управления. Впрочем, скорость прогресса в развитии вычислительной техники такова, что возможности больших ЭВМ конца 80-х годов практически по всем параметрам перекрыты наиболее мощными «супер-мини» середины 90-х. Несмотря на это, выпуск больших машин продолжается, хотя цена одной машины может составлять несколько десятков миллионов долларов.

Мини-ЭВМ появились в начале 70-х годов. Их традиционное использование -либо для управления технологическими процессами, либо в режиме разделения времени в качестве управляющей машины небольшой локальной сети. Мини-ЭВМ используются, в частности, для управления станками с ЧПУ, другим оборудованием. Среди них выделяются «супер-мини», имеющие характеристики, сравнимые с характеристиками больших машин (например, в 80-х годах таковыми считалось семейство VAX-11 фирмы DEC и его отечественные аналоги - СМ 1700 и др.).

Микро-ЭВМ обязаны своим появлением микропроцессорам. Среди них выделяют многопользовательские, оборудованные многими выносными терминалами и работающие в режиме разделения времени; встроенные, которые могут управлять станком, какой-либо подсистемой авто-

мобилия или другого устройства (в том числе и военного назначения), будучи его малой частью. Эти встроенные устройства (их часто называют контроллерами) выполняются в виде небольших плат, не имеющих рядом привычных для пользователя компьютера внешних устройств.

Термин «рабочая станция» используется в нескольких, порой несовпадающих, смыслах. Так, рабочей станцией может быть мощная микро-ЭВМ, ориентированная на специализированные работы высокого профессионального уровня, которую нельзя отнести к персональным компьютерам хотя бы в силу очень высокой стоимости. Например, это графические рабочие станции для выполнения работ по автоматизированному проектированию или для высокоуровневой издательской деятельности. Рабочей станцией могут называть и компьютер, выполняющий роль хост-машины в подузле глобальной вычислительной сети. Компьютеры фирм «Sun Microsystems», «Hewlett-Packard», стоимостью в десятки раз большей, чем персональные компьютеры, являются одно- или многопроцессорными машинами с огромным (по меркам ПК) ОЗУ, мультипроцессорной версией операционной системы, несколькими CD ROM- накопителями и т.д.

Нельзя, наконец, не сказать несколько слов об устройствах, приносящих большую пользу и также являющихся ЭВМ (поскольку они чаще всего и электронные, и вычислительные), - аналоговых вычислительных машинах (АВМ). Они уже полвека хотя и находятся на обочине развития современной вычислительной техники, но неизменно выживают. Известны системы, в которых АВМ сопрягаются с цифровыми, значительно увеличивая эффективность решения задач в целом. Основное в АВМ - они не цифровые, обрабатывают информацию, представленную не в дискретной, а в непрерывной форме (чаще всего в форме электрических токов). Их главное достоинство - способность к математическому моделированию процессов, описываемых дифференциальными уравнениями (порой очень сложных) в реальном масштабе времени. Недостаток - относительно низкая точность получаемых решений и неуниверсальность.

1.6. ЧТО ВПЕРЕДИ?

В 90-х годах микроэлектроника подошла к пределу, разрешенному физическими законами. Фантастически высока плотность упаковки компонентов в интегральных схемах и почти предельно велика возможная скорость их работы.

В совершенствовании будущих ЭВМ видны два пути. На физическом уровне это переход к использованию иных физических принципов построения узлов ЭВМ - на основе оптоэлектроники, использующей оптические свойства материалов, на базе которых создаются процессор и оперативная память, и криогенной электроники, использующей сверхпроводящие материалы при очень низких температурах. На уровне совершенствования интеллектуальных способностей машин, отнюдь не всегда определяемых физическими принципами их конструкций, постоянно возникают новые результаты, опирающиеся на принципиально новые подходы к программированию. Уже сегодня ЭВМ выигрывает шахматные партии у чемпиона мира, а ведь совсем недавно это казалось совершенно невозможным. Создание новейших информационных технологий, систем искусственного интеллекта, баз знаний, экспертных систем продолжатся в XXI веке.

Наконец, уже сегодня огромную роль играют сети ЭВМ, позволяющие разделить решение задачи между несколькими компьютерами. В недалеком будущем и сетевые технологии обработки информации станут, по-видимому, доминировать, существенно потеснив персональные компьютеры (точнее говоря, интегрировав их в себя).

В данном параграфе приведены лишь ключевые события, имена и даты в истории развития одного из наиболее замечательных технических средств, созданных человеком. Более подробную информацию можно найти в указанной в конце главы литературе.

Контрольные вопросы

1. Каковы были побудительные мотивы конструкторов первых вычислительных машин?
2. Какие ключевые события из истории развития вычислительной техники в XVII - XIX веках вам известны из дополнительной литературы?
3. Какие принципы Ч.Бэббиджа заложили в основу идеи об автоматических цифровых вычислительных машинах?
4. Почему электронная техника оказалась более подходящей для создания АЦВМ, чем ме-

ханическая и электромеханическая?

5. Каковы годы создания и названия первых ЭВМ конца 40-х - начала 50-х годов XX века?
6. Что вы знаете об истории развития отечественной вычислительной техники?
7. По каким показателям ЭВМ относят к тому или иному поколению?
8. Каковы совокупные признаки ЭВМ I-, 2-, 3-, 4-го поколений?
9. Что такое интегральная схема? большая интегральная схема?
10. Что имеют в виду, говоря о быстродействии ЭВМ? О каких операциях идет речь?
11. В чем состоят принципы пакетной обработки? разделения времени? реального времени?
12. В чем главное отличие процедурных языков высокого уровня от непроцедурных?
13. В чем состоит идея параллелизма в исполнении программ и процессов?
14. Как эволюционировало программное обеспечение общего назначения? Что входит в него сегодня?
15. В чем состоит принцип унификации линий ЭВМ?
16. В чем причины роста компьютерного парка после появления персональных ЭВМ?
17. Какие характеристики позволяют отнести компьютер к категории «персональный»?
18. Какие категории компьютеров существуют в настоящее время?
19. Что может значить термин «рабочая станция»?

§2. АРХИТЕКТУРА ЭВМ

2.1. О ПОНЯТИИ «АРХИТЕКТУРА ЭВМ»

Термин «архитектура» используется в популярной литературе по вычислительной технике достаточно часто, однако определение этого понятия и его содержание могут у разных авторов достаточно различаться. Разберемся в этом вопросе более тщательно.

Начать целесообразно с происхождения термина. Слово «архитектура» в изначальном своем смысле используется в градостроении. Будучи достаточно сложной структурой, современный город состоит из районов, площадей, улиц, домов и т.п., расположенных определенным образом. Жителей города обычно мало интересует, как выглядит конкретный дом и из каких материалов он построен. Зато очень важно знать район, где этот дом расположен, улицы, ведущие к нему, и транспорт, пользуясь которым можно сократить время в пути.

Для того, чтобы ориентироваться в хитросплетении улиц и площадей, в любом городе существует исторически сложившаяся система названий, а также определенная нумерация домов. Наличие общепринятой адресации позволяет однозначно определить положение любого строения и в случае необходимости быстро отыскать его. Именно на существовании такой адресной системы построена работа почты. Во многих случаях расположение улиц и присвоение им имен носит беспорядочный характер. В то же время бывает, что эта деятельность тщательно продумана и является продолжением общей планировки города, т.е. фактически частью его архитектуры. Классическим примером может служить известная система взаимно-перпендикулярных улиц (авеню и стриты) города Нью-Йорка. Помимо чисто практической, архитектура города может иметь еще и художественную ценность (что обычно больше интересует приезжих). Но этот аспект понятия «архитектура» вряд ли переносим на вычислительную технику.

Используя аналогию с градостроительством, естественно понимать под архитектурой ЭВМ ту совокупность их характеристик, которая необходима пользователю. Это, прежде всего, основные устройства и блоки ЭВМ, а также структура связей между ними. И действительно, если заглянуть, например, в «Толковый словарь по вычислительным системам», мы прочтем там, что термин «архитектура ЭВМ используется для описания принципа действия, конфигурации и взаимного соединения основных логических узлов ЭВМ (вследствие чего термин «архитектура» оказывается ближе к обыденному значению этого слова)».

Однако описание внутренней структуры ЭВМ вовсе не является самоцелью: с точки зрения архитектуры представляют интерес лишь те связи и принципы, которые являются наиболее общими, присущими многим конкретным реализациям вычислительных машин. Часто говорят даже о семействах ЭВМ, т.е. группах моделей, совместимых между собой. В пределах одного семейства основные принципы устройства и функционирования машин одинаковы, хотя отдельные модели могут существенно различаться по производительности, стоимости и другим параметрам. Ярким

примером могут служить различные модификации компьютеров PDP фирмы DEC (более известные нашим пользователям по отечественным аналогам - серии ДБК), семейство MSX-машин, к которому принадлежит широко распространенная YAMAHA, а также заполонившие мир IBM-совместимые персональные компьютеры.

Именно то общее, что есть в строении ЭВМ, и относят к понятию архитектуры. Важно отметить, что целью такой общности в конечном счете служит вполне понятное стремление: все машины одного семейства, независимо от их конкретного устройства и фирмы-производителя, должны быть способны выполнять одну и ту же программу (на практике из-за постоянного роста вычислительной мощности техники чаще используется менее жесткий принцип совместимости снизу вверх: все программы данной модели выполнимы на более старших). Отсюда неизбежно следует вывод, что с точки зрения архитектуры важны не все сведения о построении ЭВМ, а только те, которые могут как-то использоваться при программировании и «пользовательской» работе с ЭВМ. Равно как максимально подробная архитектура города не нуждается в описании марок кирпичей, из которых построены дома, и растворов, которыми эти кирпичи скреплены, так и архитектура ЭВМ не содержит описания электронных схем, других деталей реализации, «невидимых» для пользователя (например, внутреннего ускорителя доступа к памяти).

Ниже приводится перечень тех наиболее общих принципов построения ЭВМ, которые относятся к архитектуре:

- структура памяти ЭВМ;
- способы доступа к памяти и внешним устройствам;
- возможность изменения конфигурации компьютера;
- система команд;
- форматы данных;
- организация интерфейса.

Суммируя все вышеизложенное, получаем следующее определение архитектуры:

«Архитектура - это наиболее общие принципы построения ЭВМ, реализующие программное управление работой и взаимодействием основных ее функциональных узлов».

1.2. КЛАССИЧЕСКАЯ АРХИТЕКТУРА ЭВМ И ПРИНЦИПЫ ФОН НЕЙМАНА

Основы учения об архитектуре вычислительных машин заложил выдающийся американский математик Джон фон Нейман. Он подключился к созданию первой в мире ламповой ЭВМ ENIAC в 1944 г., когда ее конструкция была уже выбрана. В процессе работы во время многочисленных дискуссий со своими коллегами Г.Голдстайном и А.Берксом фон Нейман высказал идею принципиально новой ЭВМ. В 1946 г. ученые изложили свои принципы построения вычислительных машин в ставшей классической статье «Предварительное рассмотрение логической конструкции электронно-вычислительного устройства». С тех пор прошло полвека, но выдвинутые в ней положения сохраняют актуальность и сегодня.

В статье убедительно обосновывается использование двоичной системы для представления чисел (нелишне напомнить, что ранее все вычислительные машины хранили обрабатываемые числа в десятичном виде). Авторы убедительно продемонстрировали преимущества двоичной системы для технической реализации, удобство и простоту выполнения в ней арифметических и логических операций. В дальнейшем ЭВМ стали обрабатывать и нечисловые виды информации - текстовую, графическую, звуковую и другие, но двоичное кодирование данных по-прежнему составляет информационную основу любого современного компьютера.

Еще одной поистине революционной идеей, значение которой трудно переоценить, является предложенный Нейманом принцип «хранимой программы». Первоначально программа задавалась путем установки переключателей на специальной коммутационной панели. Это было весьма трудоемким занятием: например, для изменения программы машины ENIAC требовалось несколько дней (в то время как собственно расчет не мог продолжаться более нескольких минут - выходили из строя лампы). Нейман первым догадался, что программа может также храниться в виде набора нулей и единиц, причем в той же самой памяти, что и обрабатываемые ею числа. Отсутствие принципиальной разницы между программой и данными дало возможность ЭВМ самой формировать для себя программу в соответствии с результатами вычислений.

Фон Нейман не только выдвинул основополагающие принципы логического устройства ЭВМ, но и предложил ее структуру, которая воспроизводилась в течение первых двух поколений ЭВМ. Основными блоками по Нейману являются устройство управления (УУ) и арифметико-логическое устройство (АЛУ) (обычно объединяемые в центральный процессор), память, внешняя память, устройства ввода и вывода. Схема устройства такой ЭВМ представлена на рис. 4.10. Следует отметить, что внешняя память отличается от устройств ввода и вывода тем, что данные в нее заносятся в виде, удобном компьютеру, но недоступном для непосредственного

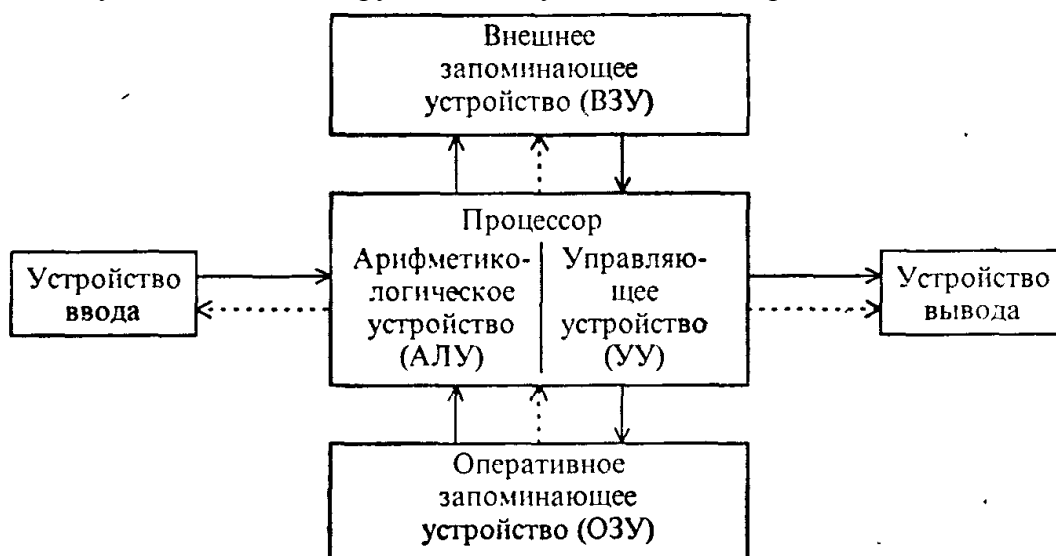


Рис. 4.10. Архитектура ЭВМ, построенной на принципах фон Неймана. Сплошные линии со стрелками указывают направление потоков информации, пунктирные-управляющих сигналов от процессора к остальным узлам ЭВМ

восприятия человеком. Так, накопитель на магнитных дисках относится к внешней памяти, а клавиатура - устройство ввода, дисплей и печать - устройства вывода.

Устройство управления и арифметико-логическое устройство в современных компьютерах объединены в один блок - процессор, являющийся преобразователем информации, поступающей из памяти и внешних устройств (сюда относятся выборка команд из памяти, кодирование и декодирование, выполнение различных, в том числе и арифметических, операций, согласование работы узлов компьютера). Более детально функции процессора будут обсуждаться ниже.

Память (ЗУ) хранит информацию (данные) и программы. Запоминающее устройство у современных компьютеров «многоярусно» и включает оперативное запоминающее устройство (ОЗУ), хранящее ту информацию, с которой компьютер работает непосредственно в данное время (исполняемая программа, часть необходимых для нее данных, некоторые управляющие программы), и внешние запоминающие устройства (ВЗУ) гораздо большей емкости, чем ОЗУ, но с существенно более медленным доступом (и значительно меньшей стоимостью в расчете на 1 байт хранимой информации). На ОЗУ и ВЗУ классификация устройств памяти не заканчивается - определенные функции выполняют и СОЗУ (сверхоперативное запоминающее устройство), и ПЗУ (постоянное запоминающее устройство), и другие подвиды компьютерной памяти.

В построенной по описанной схеме ЭВМ происходит последовательное считывание команд из памяти и их выполнение. Номер (адрес) очередной ячейки памяти, из которой будет извлечена следующая команда программы, указывается специальным устройством - счетчиком команд в УУ. Его наличие также является одним из характерных признаков рассматриваемой архитектуры.

Разработанные фон Нейманом основы архитектуры вычислительных устройств оказались настолько фундаментальными, что получили в литературе название «фон-неймановской архитектуры». Подавляющее большинство вычислительных машин на сегодняшний день - фон-неймановские машины. Исключение составляют лишь отдельные разновидности систем для параллельных вычислений, в которых отсутствует счетчик команд, не реализована классическая концепция переменной и имеются другие существенные принципиальные отличия от классической модели (примерами могут служить потоковая и редукционная вычислительные машины).

По-видимому, значительное отклонение от фон-неймановской архитектуры произойдет в

результате развития идеи машин пятого поколения, в основе обработки информации в которых лежат не вычисления, а логические выводы.

2.3. СОВЕРШЕНСТВОВАНИЕ И РАЗВИТИЕ ВНУТРЕННЕЙ СТРУКТУРЫ ЭВМ

В предыдущем разделе была описана классическая структура ЭВМ, соответствующая вычислительным машинам первого и второго поколений. Естественно, что в результате бурного развития технологии производства средств вычислительной техники такая структура не могла не претерпеть определенных прогрессивных изменений.

Как отмечалось выше, появление третьего поколения ЭВМ было обусловлено переходом от транзисторов к интегральным микросхемам. Значительные успехи в миниатюризации электронных схем не просто способствовали уменьшению размеров базовых функциональных узлов ЭВМ, но и создали предпосылки для существенного роста быстродействия процессора. Возникло существенное противоречие между высокой скоростью обработки информации внутри машины и медленной работой устройств ввода-вывода, в большинстве своем содержащих механически движущиеся части. Процессор, руководивший работой внешних устройств, значительную часть времени был бы вынужден простаивать в ожидании информации «из внешнего мира», что существенно снижало бы эффективность работы всей ЭВМ в целом. Для решения этой проблемы возникла тенденция к освобождению центрального процессора от функций обмена и к передаче их специальным электронным схемам управления работой внешних устройств. Такие схемы имели различные названия: каналы обмена, процессоры ввода-вывода, периферийные процессоры. Последнее время все чаще используется термин «контроллер внешнего устройства» (или просто контроллер).

Наличие интеллектуальных контроллеров внешних устройств стало важной отличительной чертой машин третьего и четвертого поколений.

Контроллер можно рассматривать как специализированный процессор, управляющий работой «вверенного ему» внешнего устройства по специальным встроенным программам обмена. Такой процессор имеет собственную систему команд. Например, контроллер накопителя на гибких магнитных дисках (дисковод) умеет позиционировать головку на нужную дорожку диска, читать или записывать сектор, форматировать дорожку и т.п. Результаты выполнения каждой операции заносятся во внутренние регистры памяти контроллера и могут быть в дальнейшем прочитаны центральным процессором.

Таким образом, наличие интеллектуальных внешних устройств может существенно изменять идеологию обмена. Центральный процессор при необходимости произвести обмен выдает задание на его осуществление контроллеру. Дальнейший обмен информацией может протекать под руководством контроллера без участия центрального процессора. Последний получает возможность «заниматься своим делом», т.е. выполнять программу дальше (если по данной задаче до завершения обмена ничего сделать нельзя, то можно в это время решать другую).

Перейдем теперь к обсуждению вопроса о внутренней структуре ЭВМ, содержащей интеллектуальные контроллеры, изображенной на рис. 4.11. Из рисунка видно, что для связи между отдельными функциональными узлами ЭВМ используется общая шина (часто ее называют магистралью). Шина состоит из трех частей:

- шина данных, по которой передается информация;
- шина адреса, определяющая, куда передаются данные;
- шина управления, регулирующая процесс обмена информацией.

Отметим, что существуют модели компьютеров, у которых шины данных и адреса для экономии объединены. У таких машин сначала на шину выставляется адрес, а затем через некоторое время данные; для какой именно цели используется шина в данный момент, определяется сигналами на шине управления.

Описанную схему легко пополнять новыми устройствами - это свойство называют открытостью архитектуры. Для пользователя открытая архитектура означает возможность свободно выбирать состав внешних устройств для своего компьютера, т.е. конфигурировать его в зависимости от круга решаемых задач.

На рис. 4.11 представлен новый по сравнению с рис. 4.10 вид памяти - видео-ОЗУ (видеопамять). Его появление связано с разработкой особого устройства вывода - дисплея. Основной частью дисплея служит электронно-лучевая трубка, которая отображает информацию примерно

так же, как это происходит в телевизоре (к некоторым дешевым домашним моделям компьютеров просто подключается обычный телевизор). Очевидно, что дисплей, не имея механически движущихся частей, является «очень быстрым» устройством отображения информации. Поэтому для ЭВМ третьего и четвертого поколений он является неотъемлемой частью (хотя впервые дисплей был реализован на некоторых ЭВМ второго поколения, например, на «МИР-2» - очень интересной во многих отношениях отечественной разработке).

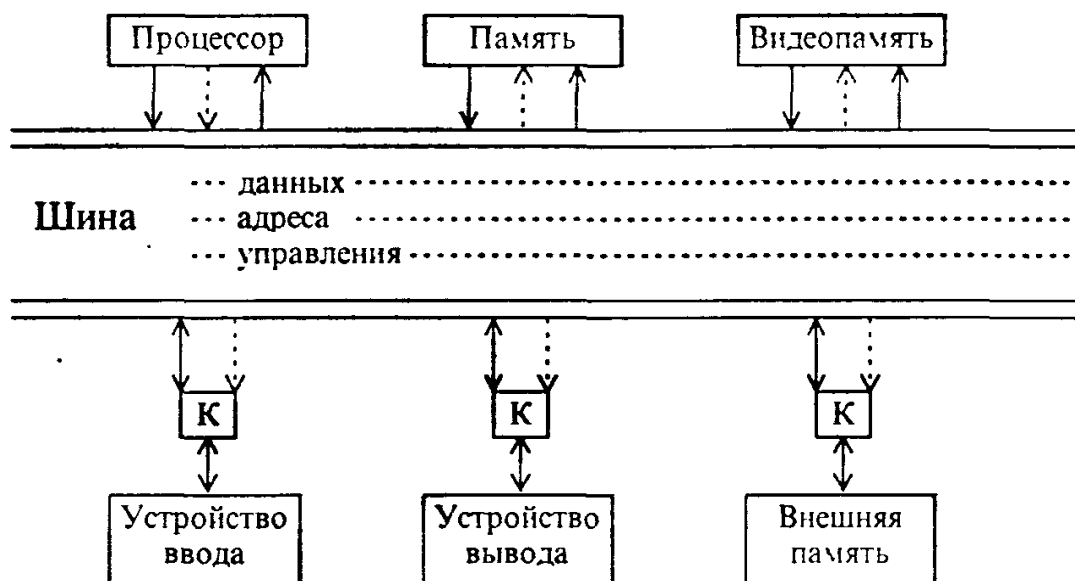


Рис. 4.11. Шинная архитектура ЭВМ

Для получения на экране монитора стабильной картинки ее надо где-то хранить. Для этого и существует видеопамять. Сначала содержимое видеопамети формируется компьютером, а затем контроллер дисплея выводит изображение на экран. Объем видеопамети существенно зависит от характера информации (текстовая или графическая) и от числа цветов изображения. Конструктивно она может быть выполнена как обычное ОЗУ или содержаться непосредственно в контроллере дисплея (именно поэтому на рис. 4.11 она показана пунктиром).

Остановимся еще на одной важной особенности структуры современных ЭВМ. Поскольку процессор теперь перестал быть центром конструкции, стало возможным реализовывать прямые связи между устройствами ЭВМ. На практике чаще всего используют передачу данных из внешних устройств в ОЗУ и наоборот. Режим, при котором внешнее устройство обменивается непосредственно с ОЗУ без участия центрального процессора, называется прямым доступом к памяти (ПДП). Для его реализации необходим специальный контроллер. Подчеркнем, что режим ПДП в машинах первого и второго поколений не существовал. Поэтому встречающаяся иногда схема ЭВМ, на которой данные из устройств ввода напрямую поступают в ОЗУ, не соответствует действительности: данные при отсутствии контроллера ПДП всегда сначала принимаются во внутренние регистры процессора и лишь затем в память.

При описании магистральной структуры мы упрощенно предполагали, что все устройства взаимодействуют через общую шину. С точки зрения архитектуры этого вполне достаточно. Упомянем все же, что на практике такая структура применяется только для ЭВМ с небольшим числом внешних устройств. При увеличении потоков информации между устройствами ЭВМ единственная магистраль перегружается, что существенно тормозит работу компьютера. Поэтому в состав ЭВМ могут вводиться одна или несколько дополнительных шин. Например, одна шина может использоваться для обмена с памятью, вторая - для связи с «быстрыми», а третья - с «медленными» внешними устройствами. Отметим, что высокоскоростная шина данных ОЗУ обязательно требуется при наличии режима ПДП.

Завершая обсуждение особенностей внутренней структуры современных ЭВМ, укажем несколько характерных тенденций в ее развитии. Во-первых, постоянно расширяется и совершенствуется набор внешних устройств, что приводит, как описывалось выше, к усложнению системы связей между узлами ЭВМ. Во-вторых, вычислительные машины перестают быть однопроцессорными. Помимо центрального, в компьютере могут быть специализированные процессоры для вычисления с плавающей запятой (так называемые математические сопроцессоры), видеопроцессоры

для ускорения вывода информации на экран дисплея и т.п. Развитие методов параллельных вычислений также вызывает к жизни вычислительные системы достаточно сложной структуры, в которых одна операция выполняется сразу несколькими процессорами. В-третьих, наметившееся стремление иметь быстродействующие машины не только для вычислений, но и для логического анализа информации, также может привести в ближайшие годы к серьезному пересмотру традиционной фон-неймановской архитектуры.

Еще одной особенностью развития современных ЭВМ является все ускоряющееся возрастание роли межкомпьютерных коммуникаций. Все большее количество компьютеров объединяются в сети и обрабатывают имеющуюся информацию совместно.

Таким образом, внутренняя структура вычислительной техники постоянно совершенствовалась и будет совершенствоваться. Вместе с тем, на данный момент подавляющее большинство существующих ЭВМ, несмотря на имеющиеся различия, по-прежнему состоит из одинаковых узлов и основано на общих принципах фон-неймановской архитектуры.

2.4. ОСНОВНОЙ ЦИКЛ РАБОТЫ ЭВМ

В данном разделе коротко рассмотрена последовательность действий при выполнении команды в ЭВМ. Можно утверждать, что рабочий цикл в общем виде одинаков для всех фон-неймановских машин.

Как уже отмечалось в п.2.2, важной составной частью фон-неймановской архитектуры является счетчик адреса команд. Этот специальный внутренний регистр процессора всегда указывает на ячейку памяти, в которой хранится следующая команда программы. При включении питания или при нажатии на кнопку сброса (начальной установки) в счетчик аппаратно заносится стартовый адрес находящейся в ПЗУ программы инициализации всех устройств и начальной загрузки. Дальнейшее функционирование компьютера определяется программой. Таким образом, вся деятельность ЭВМ - это непрерывное выполнение тех или иных программ, причем программы эти могут в свою очередь загружать новые программы и т.д.

Каждая программа состоит из отдельных машинных команд. Каждая машинная команда, в свою очередь, делится на ряд элементарных унифицированных составных частей, которые принято называть тактами. В зависимости от сложности команды она может быть реализована за разное число тактов. Например, пересылка информации из одного внутреннего регистра процессора в другой выполняется за несколько тактов, а для перемножения двух целых чисел их требуется на порядок больше. Существенное удлинение команды происходит, если обрабатываемые данные еще не находятся внутри процессора и их приходится считывать из ОЗУ.

При выполнении каждой команды ЭВМ выполняет определенные стандартные действия:

1) согласно содержимому счетчика адреса команд, считывается очередная команда программы (ее код обычно заносится на хранение в специальный регистр УУ, который носит название регистра команд);

2) счетчик команд автоматически изменяется так, чтобы в нем содержался адрес следующей команды (в простейшем случае для этой цели достаточно к текущему значению счетчика прибавить некоторую константу, определяющуюся длиной команды);

3) считанная в регистр команд операция расшифровывается, извлекаются необходимые данные и над ними выполняются требуемые действия.

Затем во всех случаях, за исключением команды останова или наступления прерывания (см. ниже в п. 3.5), все описанные действия циклически повторяются.

После выборки команды останова ЭВМ прекращает обработку программы. Для выхода из этого состояния требуется либо запрос от внешних устройств, либо перезапуск машины.

Рассмотренный основной алгоритм работы ЭВМ позволяет шаг за шагом выполнить хранящуюся в ОЗУ линейную программу. Если же требуется изменить порядок вычислений для реализации развилки или цикла, достаточно в счетчик команд занести требуемый адрес (именно так происходит условный или безусловный переход).

В компьютерах на базе микропроцессоров INTEL 80286 и более поздних моделей для ускорения основного цикла выполнения команды используется метод конвейеризации (иногда применяется термин «опережающая выборка»). Идея состоит в том, что несколько внутренних устройств процессора работают параллельно: одно считывает команду, другое дешифрует опера-

цию, третье вычисляет адреса используемых операндов и т.д. В результате по окончании команды чаще всего оказывается, что следующая уже выбрана из ОЗУ, дешифрована и подготовлена к исполнению. Отметим, что в случае нарушения естественного порядка выполнения команд в программе (например, при безусловном переходе) опережающая выборка оказывается напрасной и конвейер очищается. Следующая за переходом команда выполняется дольше, так как, чтобы конвейер «заработал на полную мощность», необходимо его предварительно заполнить. Иными словами, в конвейерной машине время выполнения программы может зависеть не только от составляющих ее команд, но и от их взаимного расположения.

2.5. СИСТЕМА КОМАНД ЭВМ И СПОСОБЫ ОБРАЩЕНИЯ К ДАННЫМ

Важной составной частью архитектуры ЭВМ является система команд. Несмотря на большое число разновидностей ЭВМ, на самом низком («машинном») уровне они имеют много общего. Система команд любой ЭВМ обязательно содержит следующие группы команд обработки информации.

1. Команды передачи данных (перепись), копирующие информацию из одного места в другое.

2. Арифметические операции, которым фактически обязана своим названием вычислительная техника. Конечно, доля вычислительных действий в современном компьютере заметно уменьшилась, но они по-прежнему играют в программах важную роль. Отметим, что к основным арифметическим действиям обычно относятся сложение и вычитание (последнее в конечном счете чаще всего тем или иным способом также сводится к сложению). Что касается умножения и деления, то они во многих ЭВМ выполняются по специальным программам.

3. Логические операции, позволяющие компьютеру анализировать обрабатываемую информацию. Простейшими примерами могут служить сравнение, а также известные логические операции И, ИЛИ, НЕ (инверсия). Кроме того к ним часто добавляются анализ отдельных битов кода, их сброс и установка.

4. Сдвиги двоичного кода влево и вправо. Для доказательства важности этой группы команд достаточно вспомнить правило умножения столбиком: каждое последующее произведение записывается в такой схеме со сдвигом на одну цифру влево. В некоторых частных случаях умножение и деление вообще может быть заменено сдвигом (вспомните, что дописав или убрав ноль справа, т.е. фактически осуществляя сдвиг десятичного числа, можно увеличить или уменьшить его в 10 раз).

5. Команды ввода и вывода информации для обмена с внешними устройствами. В некоторых ЭВМ внешние устройства являются специальными служебными адресами памяти, поэтому ввод и вывод осуществляется с помощью команд переписи.

6. Команды управления, реализующие нелинейные алгоритмы. Сюда прежде всего следует отнести условный и безусловный переход, а также команды обращения к подпрограмме (переход с возвратом). Некоторые ЭВМ имеют специальные команды для организации циклов, но это не обязательно: цикл может быть сведен к той или иной комбинации условного и безусловного переходов. Часто к этой же группе команд относят немногочисленные операции по управлению процессором - типа «останов» или НОП («нет операции»). Иногда их выделяют в особую группу.

С ростом сложности устройства процессора увеличивается и число команд, анализирующих состояние управляющих битов и воздействующих на них. Здесь для примера можно назвать биты режима работы процессора и биты управления механизмами прерываний от внешних устройств.

В последнее время все большую роль в наборе команд играют команды для преобразования из одного формата данных в другой (например, из 8-битного в 16-битный и т.п.), которые заметно упрощают обработку данных разного типа, но в принципе могут быть заменены последовательностью из нескольких более простых команд.

Рассматривая систему команд, нельзя не упомянуть о двух современных взаимно конкурирующих направлениях в ее построении: компьютер с полным набором команд CISC (Complex Instruction Set Computer) и с ограниченным набором - RISC (Reduced Instruction Set Computer). Разделение возникло из-за того, что основную часть времени компьютеру приходится выполнять небольшую часть из своего набора команд, остальные же используются эпизодически (в одной из

популярных статей это в шутку сформулировано в виде следующей наглядной аналогии: «20% населения выпивают 80% пива»). Таким образом, если существенно ограничить набор операций до наиболее простых и коротких, зато тщательно оптимизировать их, получится достаточно эффективная и быстродействующая RISC-машина. Правда за скорость придется платить необходимостью программной реализации «отброшенных» команд, но часто эта плата бывает оправданной: например, для научных расчетов или машинной графики быстродействие существенно важнее проблем программирования. Подробнее вопросы, связанные с системой команд современных микропроцессоров, будут рассмотрены ниже в этой главе.

Подводя итог, еще раз подчеркнем, что основной набор команд довольно слабо изменился в ходе бурной эволюции ЭВМ. В то же время способы указания адреса расположения информации в памяти претерпели значительное изменение и заслуживают особого рассмотрения.

Команда ЭВМ обычно состоит из двух частей - операционной и адресной. Операционная часть (иначе она еще называется кодом операции - КОП) указывает, какое действие необходимо выполнить с информацией. Адресная часть описывает, где используемая информация хранится. У нескольких немногочисленных команд управления работой машины адресная часть может отсутствовать, например, в команде останова; операционная часть имеется, всегда.

Код операции можно представить себе как некоторый условный номер в общем списке системы команд. В основном этот список построен в соответствии с определенными внутренними закономерностями, хотя они не всегда очевидны.

Адресная часть обладает значительно большим разнообразием и ее следует рассмотреть подробнее.

Прежде всего отметим, что команды могут быть одно-, двух- и трехадресные в зависимости от числа участвующих в них операндов.

Первые ЭВМ имели наиболее простую и наглядную трехадресную систему команд. Например: взять числа из адресов памяти A1 и A2, сложить их и сумму поместить в адрес A3. Если для операции требовалось меньшее число адресов, то лишние просто не использовались. Скажем, в операции переписи указывались лишь ячейки источника и приемника информации A1 и A3, а содержимое A2 не имело никакого значения.

Трехадресная команда легко расшифровывалась и была удобна в использовании, но с ростом объемов ОЗУ ее длина становилась непомерно большой. Действительно, длина команды складывается из длины трех адресов и кода операции. Отсюда следует, например, что для скромного ОЗУ из 1024 ячеек только для записи адресной части одной команды требуется $3 \cdot 10 = 30$ двоичных разрядов, что для технической реализации не очень удобно. Поэтому появились двухадресные машины, длина команды в которых сокращалась за счет исключения адреса записи результата. В таких ЭВМ результат операции оставался в специальном регистре (сумматоре) и был пригоден для использования в последующих вычислениях. В некоторых машинах результат записывался вместо одного из операндов.

Дальнейшее упрощение команды привело к созданию одноадресных машин. Рассмотрим систему команд такой ЭВМ на конкретном простом примере. Пусть надо сложить числа, хранящиеся в ячейках с адресами ОЗУ A1 и A2, а сумму поместить в ячейку с адресом A3. Для решения этой задачи одноадресной машине потребуется выполнить три команды:

- извлечь содержимое ячейки A1 в сумматор;
- сложить сумматор с числом из A2;
- записать результат из сумматора в A3.

Может показаться, что одноадресной машине для решения задачи потребуется втрое больше команд, чем трехадресной. На самом деле это не всегда так. Попробуйте самостоятельно спланировать программу вычисления выражения $A5 = (A1 + A2) \cdot A3 / A4$ и вы обнаружите, что требуется три трехадресные команды и всего пять одноадресных. Таким образом, одноадресная машина в чем-то даже эффективнее, так как она не производит ненужной записи в память промежуточных результатов.

Ради полноты изложения следует сказать о возможности реализации безадресной (нуль-адресной) машины, использующей особый способ организации памяти - стек. Понимание принципов устройства такой машины потребовало бы некоторых достаточно подробных разъяснений. Сейчас безадресные ЭВМ практически не применяются. Поэтому ограничимся лишь упоминанием того факта, что устроенная подобным образом система команд лежала в основе некоторых про-

граммируемых микрокалькуляторов (например, типа «БЗ-21» и «БЗ-34» и им подобных).

До сих пор в описании структуры машинной команды мы пользовались интуитивным понятием об адресе информации. Рассмотрим теперь вопрос об адресации элементов ОЗУ более подробно и строго. Наиболее просто была организована память в ЭВМ первых двух поколений. Она состояла из отдельных ячеек, содержимое каждой из которых считывалось или записывалось как единое целое. Каждая ячейка памяти имела свой номер, который и получил название адреса. Очевидно, что адреса соседних ячеек ОЗУ являются последовательными целыми числами, т.е. отличаются на единицу. В рассматриваемых ЭВМ использовались данные только одного типа (вещественные числа), причем их длина равнялась длине машинной команды и совпадала с разрядностью памяти и всех остальных устройств машины. Для примера укажем, что ячейка типичной ЭВМ второго поколения состояла из 36 двоичных разрядов.

Очень часто программа предназначалась для обработки по одним и тем же формулам определенного количества содержимого последовательно расположенных ячеек (в языках высокого уровня такого рода структуры получили впоследствии название массивов). В ЭВМ первых двух поколений были предусмотрены особые механизмы циклической обработки массивов информации. С этой целью в машинных командах помимо обычных адресов можно было использовать модифицируемые, у которых специальный управляющий бит был установлен в единицу. К помеченным таким образом модифицируемым адресам при выполнении команды прибавлялось значение из специальных индексных ячеек. Меняя содержимое индексных ячеек, можно было получать доступ к различным элементам массива. Особо подчеркнем, что формирование результирующего адреса осуществлялось в УУ в момент исполнения команды, поэтому исходная команда в ОЗУ сохранялась без изменений.

Описанный механизм модификации адресов существенно упрощал написание циклических программ, таких как нахождение суммы последовательных ячеек ОЗУ, копирование отдельных участков памяти и т.п.

В ЭВМ третьего поколения идеология построения памяти существенно изменилась: минимальная порция информации для обмена с ОЗУ была установлена равной 8 двоичных разрядов, т.е. один байт. Стало возможным обрабатывать несколько типов данных: символы текста (1 байт), целые числа (2 байта), вещественные числа обычной или двойной точности (4 или 8 байт соответственно). В связи с этим была введена новая условная единица измерения информации - машинное слово. Оно равнялось 4 байтам и соответствовало длине стандартного вещественного числа. Все объемы информации начали измеряться в единицах, кратных слову: двойное слово, полуслово и т.п. Естественно, что адрес (номер ячейки ОЗУ) в машинах с байтовой организацией стал относиться к отдельному байту; байты памяти имеют возрастающие на единицу номера. Слово состоит из нескольких последовательно расположенных байтов. В качестве адреса слова удобно принимать адрес одного из образующих его байтов (обычно используется младший байт, имеющий наименьший номер). Таким образом, адреса слов меняются уже не через единицу; их приращение зависит от длины машинного слова в байтах и равняется четырем.

Размер машинного слова был, по-видимому, выбран исходя из форматов обрабатываемой информации, а не в связи с разрядностью каких-либо устройств. Для подтверждения этого приведем несколько фактов о типичных ЭВМ третьего поколения из семейства ЕС. Арифметико-логическое устройство модели «ЕС-1022» имело 16 двоичных разрядов, «ЕС-1033» - 32 разряда, а «ЕС-1050» - 64 разряда. В то же время за одно обращение к оперативной памяти в «ЕС-1022» и «ЕС-1033» выбиралось 4 байта, в «ЕС-1050» - 8 байт (а в «ЕС-1045» - 16 байт). Таким образом, разнообразие цифр свидетельствует, что 32 разряда (4 байта) не являлись каким-то технически выделенным объемом информации.

В машинах третьего поколения появились и еще несколько особенностей: разная длина команд в зависимости от способа адресации данных, наличие специальной сверхоперативной регистровой памяти, вычисление эффективного адреса ОЗУ как суммы нескольких регистров и т.п. Все это получило дальнейшее развитие в компьютерах четвертого поколения, для которых разрядность микропроцессора стала одной из важнейших характеристик. Рассмотрение особенностей строения памяти ЭВМ четвертого поколения отложим до следующего раздела.

Контрольные вопросы

1. Что такое архитектура ЭВМ? Сформулируйте определение и расшифруйте его.

2. Проведите аналогию между архитектурой ЭВМ и обыденным понятием архитектуры. Что общего и в чем различие?
3. Что общего и в чем различие между понятиями «внутреннее устройство ЭВМ» и «архитектура ЭВМ»?
4. Что такое семейство ЭВМ? Приведите примеры.
5. Объясните, в чем состоит принцип программной совместимости. Что такое совместимость снизу вверх (поясните на примере одного из известных вам семейств)?
6. Имеют ли отношение к понятию «архитектура» следующие факты:
- а) в компьютере применяются микросхемы динамического (или статического) ОЗУ?
 - б) компьютер имеет расширенную память?
 - в) компьютер имеет (не имеет) общую шину, по которой передается информация между его устройствами?
 - г) в процессоре INTEL 80386 к системе команд добавлено по сравнению с INTEL 80286 несколько новых?
 - д) объем памяти новой модели ЭВМ увеличен вдвое?
7. Перечислите основные принципы фон-неимановской архитектуры и разъясните их содержание.
8. Чем обусловлено в ЭВМ широкое применение двоичной системы?
9. Можно ли, посмотрев на содержимое отдельно взятой ячейки памяти, определить, какая информация в ней записана: число, команда, символы?
10. Из каких основных элементов состоит ЭВМ?
11. Что такое счетчик команд и какую роль он играет?
12. Что такое магистраль (шина)?
13. Какие преимущества имеет магистральная структура ЭВМ?
14. Что представляет собой контроллер внешнего устройства и какую роль он играет в процессе обмена информацией?
15. Какую роль играет в компьютере видеопамять?
16. Оцените необходимый объем видеопамяти для следующих режимов:
- а) текстовый режим (24 строки по 80 символов);
 - б) графический черно-белый режим при размере экрана 640x200 точек;
 - в) 16-цветный режим при том же размере экрана.
17. Что такое режим прямого доступа к памяти?
18. Как называется элементарная составляющая машинной команды? От чего может зависеть скорость выполнения команды?
19. Опишите основные этапы выполнения машинной команды. Особое внимание обратите на роль счетчика команд.
20. Что такое конвейерная обработка команд и какие преимущества она имеет?
21. Какие основные операции входят в состав системы команд любой ЭВМ? Кратко охарактеризуйте каждую из названных групп.
22. Объясните, почему возможно создать компьютер с уменьшенным (неполным) набором команд и что это дает.
23. Из каких частей состоит команда ЭВМ? Кратко охарактеризуйте их назначение.
24. Чем различаются одно-, двух- и трехадресные команды?
25. Что такое адрес ОЗУ?
26. Как можно использовать одну и ту же команду для работы с несколькими последовательно расположенными ячейками?
27. Укажите отличия в устройстве памяти ЭВМ третьего поколения по сравнению с двумя предыдущими.

§3. АРХИТЕКТУРА МИКРОПРОЦЕССОРОВ

3.1. ИСТОРИЯ РАЗВИТИЯ МИКРОПРОЦЕССОРОВ

В 1959 г. инженеры фирмы «Texas Instruments» разработали способ, как разместить внутри одного полупроводникового кристалла несколько транзисторов и соединить их между собой - ро-

дилась первая интегральная микросхема (ИМС). По сравнению с функционально теми же устройствами, собранными из отдельных транзисторов, резисторов и т.п.. ИМС обладает значительными преимуществами: меньшими габаритами, более высокой надежностью и т.д. Неудивительно, что количество выпускаемых микросхем стало быстро возрастать, а их ассортимент неуклонно расширяться. Последнее обстоятельство создавало ряд трудностей для потребителей. Важно даже не столько то, что стремительно возраставшее количество типов ИМС затрудняло ориентацию в море наименований. Значительно большим недостатком была узкая специализация ИМС, из-за которой объем их выпуска не мог быть большим, а значит стоимость одной микросхемы оставалась высокой. Улучшить ситуацию позволило бы создание универсальной логической ИМС, специализация которой определялась бы не заложенной на заводе внутренней структурой, а заданной непосредственно самим потребителем программой работы.

Таким образом, оказывается, что первые микропроцессоры (МП) появились совсем не для миниатюризации ЭВМ, а в целях создания более дешевой логической микросхемы, легко адаптируемой к потребностям пользователя.

История создания первого в мире микропроцессора достаточно поучительна. Летом 1969 г. японская компания «Busicom», разрабатывавшая новое семейство калькуляторов, обратилась за помощью в фирму «Intel». К тому времени «Intel» просуществовала всего около года, но уже проявила себя созданием самой емкой на тот момент микросхемы памяти. Фирме «Busicom» как раз и требовалось изготовить микросхемы, содержащие несколько тысяч транзисторов. Для реализации совместного проекта был привлечен инженер фирмы «Intel» М.Хофф. Он познакомился с разработками «Busicom» и предложил альтернативную идею: вместо 12 сложных специализированных микросхем создать одну программируемую универсальную - микропроцессор. Проект Хоффа победил и фирма «Intel» получила контракт на производство первого в мире микропроцессора.

Практическая реализация идеи оказалась непростым делом. В начале 1970 г. к работе подключился Ф.Фаджин, который за 9 месяцев довел процессор от описания до кристалла (позднее Ф.Фаджин основал фирму «Zilog», создавшую замечательный 8-разрядный процессор Z80, который и сейчас успешно работает во многих домашних компьютерах). 15 ноября 1971 г. «Intel 4004» - так называли процессор - был представлен общественности.

Поскольку для хранения одной цифры калькулятору требуется 4 бита (именно столько необходимо для изображения десятичных цифр «8» и «9»), «Intel 4004» был четырехразрядным процессором. Следующий микропроцессор предназначался для установки в терминал и должен был обрабатывать символьную информацию. Поскольку каждый символ кодируется одним байтом, следующая модель «Intel 8008» стала 8-разрядной; она появилась в апреле 1982 г. По-прежнему этот процессор был заменой «аппаратной логики», но отдельные энтузиасты уже пытались собрать на нем компьютер. Результаты были скорее демонстрационными, нежели полезными, но микрокомпьютерная революция уже началась.

А в апреле 1974 г. компания «Intel» совершила новый качественный скачок: ее изделие с маркой «Intel 8080» стало первым в мире процессором, походившим на «настоящую» вычислительную машину. Отметим любопытную деталь: хотя процессор и обрабатывал 8-разрядные данные, но адрес ОЗУ был двухбайтовым! Таким образом, 8080 мог иметь до 64 килобайт памяти, что по тем временам казалось программистам недостижимым пределом.

Дальнейшее развитие событий происходило прямо-таки с фантастической скоростью, даже если сравнивать с темпами динамично развивающейся вычислительной техники. За десятилетие был пройден путь от изобретения 4-разрядного МП до достаточно сложной 32-разрядной архитектуры. Было ликвидировано отставание микропроцессорной техники от обычных ЭВМ и началось интенсивное вытеснение последних (все ЭВМ четвертого поколения собраны на базе того или иного микропроцессора!). Для иллюстрации укажем, что первый МП 4004 содержал 2200 транзисторов, МП 8080 - 4800, МП «Intel 80486» - около 1,2 миллиона, а современный «Pentium» - около 3 миллионов!

История развития микропроцессоров представляет собой достаточно интересную самостоятельную тему. Здесь упомянем только, что пионер в создании процессорных микросхем фирма «Intel» по-прежнему сохраняет свои лидирующие позиции в этой области. Ее программно-совместимое семейство последовательно усложняющихся МП (16-разрядные 8086, 80286 и 32-разрядные 80386, 80486, «Pentium») являются «мозгом» значительной части использующихся компьютеров. Именно на базе этих микропроцессоров собраны все широко распространенные в

нашей стране IBM-совместимые компьютеры.

Другую ветку обширного микропроцессорного семейства образуют МП фирмы «Motorola»: ее изделия работают в известных компьютерах «Apple», а также в более простых - «Atari», «Com-mador», «Amiga» и др. Процессоры «Motorola» ничуть не хуже, а порой даже заметно лучше производимых компанией «Intel». Но на стороне последней - огромные производственные мощности транснационального гиганта IBM и десятки южно-азиатских фирм, буквально наводнившие мир дешевыми IBM-совместимыми компьютерами.

В 1993 г. фирма «Motorola» совместно с IBM и «Apple» разработала новый процессор «PowerPC». Этот процессор имеет очень хорошие технические характеристики, но самое главное в нем - он может эмулировать работу компьютеров и «Apple», и IBM. Очевидно, что это событие еще более обострит конкурентную борьбу на рынке микропроцессоров.

Завершая краткий исторический экскурс, попробуем определить некоторые новые направления развития МП в ближайшем будущем. Характерной чертой последних моделей процессоров является возможность работы в многозадачном режиме, который фактически стал нормой для современных ЭВМ. Развивается RISC-архитектура микропроцессоров (процессоры с минимальным числом команд). Такой МП работает необычайно быстро и способен выполнить любую из своих немногочисленных команд за один машинный такт, в то время как обычно на выполнение простой операции требуется 4-5 тактов. Ярким примером достоинств RISC-архитектуры является уже упоминавшийся процессор «PowerPC». Следует особо подчеркнуть, что успехи RISC-подхода оказывают существенное влияние и на конструирование CISC-процессоров (процессоры с полным набором команд). Так, существенное ускорение классических CISC МП старших моделей семейства «Intel» достигается за счет конвейерного выполнения команд, заимствованного из RISC МП.

И, наконец, нельзя не упомянуть о транспьютерах, содержащих в процессорном кристалле собственное ОЗУ от 2 до 16 кбайт и каналы связи с внешним ОЗУ и с другими транспьютерами. Теоретические возможности этих ИМС, реализующих алгоритмы параллельных вычислений, поражают воображение. Однако потребуется значительное время, прежде чем они смогут быть практически реализованы.

Не следует думать, что бурное развитие микропроцессоров требуется только для вычислительных машин, где МП используются уже не только в качестве центрального процессора, но и в качестве контроллеров для управления сложными периферийными устройствами типа винчестера или лазерного принтера. Все большее число ИМС ставится в изделия, напрямую не связанные с ЭВМ, в том числе и бытовые: лазерные аудио- и видеопроекторы, телетекст и пейджинговая связь, программируемые микроволновые печи и стиральные машины, а также многие другие. Очевидно, что число таких управляемых микропроцессорами устройств будет все время возрастать.

3.3. ВНУТРЕННЯЯ ОРГАНИЗАЦИЯ МИКРОПРОЦЕССОРА

Перечислим основные функции микропроцессора:

- выборка команд из ОЗУ;
- декодирование команд (т.е. определение назначения команды, способа ее исполнения и адресов операндов);
- выполнение операций, закодированных в командах;
- управление пересылкой информации между своими внутренними регистрами, оперативной памятью и внешними (периферийными) устройствами;
- обработка внутрипроцессорных и программных прерываний;
- обработка сигналов от внешних устройств и реализация соответствующих прерываний;
- управление различными устройствами, входящими в состав компьютера.

Внутреннее устройство микропроцессоров очень сложно (вспомним три миллиона транзисторов в «Pentium»). Даже если попытаться рассмотреть наиболее общую схему основных функциональных узлов, и то получится достаточно сложная картина. К тому же внутреннее устройство МП сильно зависит от его марки, а стало быть изучение структуры одного процессора не обязательно помогает понять работу другого. Следует признать нецелесообразным для пользователя (и даже, может быть, для программиста) изучение инженерных деталей процессора современной ЭВМ, и ограничиться, как это принято делать, только теми функциональными узлами, которые доступны программно. При таком подходе оказывается, что МП имеют много общего, и становят-

ся отчетливо видны некоторые закономерности их внутреннего устройства. Кроме того, исчезает пугающая сложность и возникает приятное и полезное чувство, что компьютер - это не какая-то там «вещь в себе» и его поведение можно понять.

Итак, что же представляет собой микропроцессор с точки зрения программиста? Рассмотрение начнем в наиболее общем виде, не конкретизируя пока тип МП.

Ответ на поставленный вопрос, как ни странно, будет чрезвычайно прост: для программиста любой процессор состоит из набора регистров памяти различного назначения, которые определенным образом связаны между собой и обрабатываются в соответствии с некоторой системой правил. Конечно, программисту доступна не вся внутренняя память процессора: есть множество рабочих (программно-недоступных) регистров, использующихся только во время выполнения команд и т.п.; их мы рассматривать не будем.

Обсуждение внутренних регистров микропроцессора начнем с наиболее важных: *счетчика адреса команд*, *указателя стека* и *регистра состояния*. Наличие счетчика адреса команд, как уже говорилось выше, было предложено еще в работах фон Неймана. Роль *счетчика* состоит в сохранении адреса очередной команды программы и автоматическом вычислении адреса следующей. Благодаря наличию программного счетчика в ЭВМ реализуется основной цикл исполнения последовательно расположенных команд программы. Заметим, что не во всех МП счетчик команд программно доступен.

В *указателе стека* хранится адрес начала специальным образом организуемого участка памяти стека, описанного в следующем параграфе; роль указателя стека в функционировании процессора достаточно велика.

Наконец, *регистр состояния процессора*. Для разных МП он может называться по-разному (например, слово состояния процессора, регистр флагов и т.п.), но суть его всегда одна: в этом регистре хранятся сведения о текущих режимах работы процессора. Сюда же помещается информация о результатах выполняемых команд, например, равен ли результат нулю, отрицателен ли он, не возникли ли ошибки в ходе операции и т.п. Использование и анализ информации в этом очень важном регистре происходит побитно; иными словами, каждый бит регистра состояния имеет самостоятельное значение. Содержание регистра состояния МП всегда старается сохранить в первую очередь сразу после значения командного счетчика.

Помимо рассмотренных выше, каждый МП имеет набор *рабочих регистров*, в которых хранятся текущие обрабатываемые данные или их адреса в ОЗУ. У некоторых процессоров регистры функционально равнозначны (классическим примером служит процессор машин семейства PDP), в других (к ним принадлежит все интел-ловское семейство МП) назначение регистров достаточно жестко оговаривается. В последнем случае выделяется особый регистр, который принято называть *аккумулятором*. В нем производятся все основные операции и сохраняется их результат

Завершая разговор о регистрах, укажем на существование определенных связей между ними: информация из одного может передаваться в другой. Для машин с равноправными регистрами передача данных возможна между любыми регистрами, для остальных - между строго определенными парами. Так или иначе (в худшем случае за несколько машинных команд) информацию из одного регистра МП всегда можно перенести в другой.

Разрядность регистров МП существенно влияет на возможности всей ЭВМ. Поэтому уточним понятие «разрядность ЭВМ». Оно включает

- разрядность внутренних регистров микропроцессора (m);
- разрядность шины данных (n);
- разрядность шины адреса (k).

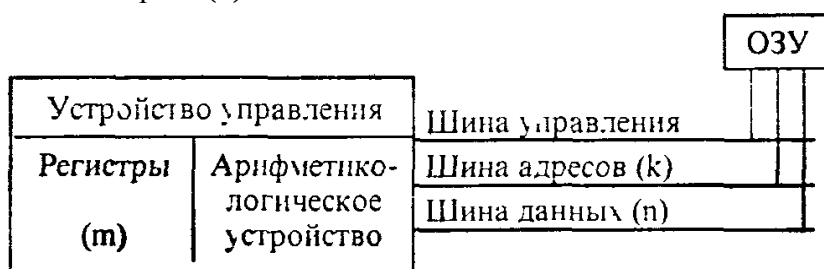


Рис. 4.12. Обмен информацией между процессором и основной памятью

Схема обмена информацией между микропроцессором и ОЗУ представлена на рис. 4.12. Данные поступают из ОЗУ в регистры процессора и наоборот по шине данных; по шине адреса передается информация о месте хранения данных в ОЗУ. Разрядности этих шин могут не совпадать: например, упомянутый выше МП «Intel 8088» характеризовался значениями $m/n/k = 16/8/20$. Когда говорят, не вникая в детали, «16-разрядная ЭВМ», то имеют в виду значение n . Поскольку объем адресного пространства ОЗУ, контролируемого МП, равен $2k$, то понятно стремление увеличить разрядность шины адреса.

3.3. РАБОТА МИКРОПРОЦЕССОРА С ПАМЯТЬЮ. МЕТОДЫ АДРЕСАЦИИ

Адресное пространство МП состоит из множества ячеек памяти ОЗУ, из которых он может брать информацию или засылать ее. Как говорилось выше, начиная с 4-го поколения доминирует байтовая организация памяти, и минимально адресуемой единицей является байт. Например, для ОЗУ емкостью 1024 кбайта адреса байтов таковы:

00000 00001 ... FFFFF

(как принято, записываем их в шестнадцатеричной системе; адрес последнего байта есть $1024 \cdot 1024 - 1 = 1\,048\,576 - 1 = \text{FFFFF}$). Длина же ячейки («машинного слова») может быть как один, так и несколько байтов в зависимости от типа процессора и команды, обрабатывающей соответствующую информацию.

При обмене информацией с памятью процессор обращается к ячейкам ОЗУ по их номерам (адресам). Способы задания требуемых адресов в командах ЭВМ принято называть *методами адресации*. От видов и разнообразия методов адресации существенно зависит эффективность работы программы с данными, особенно если последние организованы в определенную структуру.

Для того, чтобы процессор мог извлечь данные из ячейки ОЗУ или поместить их туда, необходимо где-то задать требуемый адрес. Если адрес находится в самой команде, то мы имеем дело с *прямой адресацией*. Поскольку при подобном способе слишком сильно возрастает длина команды, то, чтобы избежать этого неприятного эффекта, при обращении к ОЗУ процессор использует метод *косвенной адресации*. Идея состоит в том, что адрес памяти предварительно заносится в один из регистров МП, а в команде содержится лишь ссылка на этот регистр. Если учесть, что при хранении адреса в регистре его еще очень удобно модифицировать (скажем, циклически увеличивая на заданную величину), становится понятным, почему косвенная адресация нашла такое широкое применение.

Приведем описание наиболее распространенных вариантов ссылок на исходную информацию (учитывая, что терминология для разных МП может различаться, названия методов адресации не приводятся):

- 1) данные находятся в одном из регистров МП;
- 2) данные входят непосредственно в состав команды, т.е. размещаются после кода операции (операции с константой);
- 3) данные находятся в ячейке ОЗУ, адрес которой содержится в одном из регистров МП;
- 4) данные находятся в ячейке ОЗУ, адрес которой вычисляется по формуле

$$\text{адрес} = \text{базовый адрес} + \text{смещение}.$$

Базовый адрес хранится в одном из регистров МП и является начальной точкой массива данных. Смещение может быть как некоторой константой, так и содержимым другого регистра. Часто такой способ доступа к ОЗУ называют *индексным*, так как это похоже на нахождение элемента в одномерном массиве по его индексу.

Следует подчеркнуть, что здесь описаны лишь наиболее общие методы адресации. У конкретных моделей МП существуют некоторые особенности адресации ОЗУ. Кроме того, имеющиеся методы адресации могут быть комбинированными. Так, например, в процессорах семейства RDP возможна двойная косвенная адресация: данные хранятся в ячейке ОЗУ, адрес которой хранится в ячейке, адрес которой находится в указанном регистре (как тут не вспомнить детский стишок о синице, которая ворует пшеницу, которая в темном чулане хранится, в доме, который построил Джек?).

Методы адресации могут быть и более экзотическими. Рассмотрим, например, широко распространенный *сегментный* способ, принятый в процессорах фирмы «Intel». Известный американ-

ский программист Питер Нортон метко назвал этот способ словом «клудж» (от английского «kludge» - приспособление для временного устранения проблемы). Сегментный метод адресации был предложен для первого 16-разрядного МП 8086 для того, чтобы, используя 16-разрядные регистры, можно было получить 20-разрядный адрес и тем самым расширить максимально возможный объем ОЗУ с 64 кбайт до 1 Мбайта. Суть метода состоит в том, что адрес ОЗУ вычисляется как сумма двух чисел (сегмента и смещения), причем одно из них сдвинуто влево на 4 двоичных разряда, т.е. умножено на 16. Пусть, например, сегмент в шестнадцатеричном виде равен A000, а смещение - 1000. Общепринятая запись такого адреса имеет вид A000: 1000. Итоговый адрес равен

$$\begin{array}{r} A000() \\ + \quad 1000 \\ \hline A1000 \end{array}$$

«Такой способ адресации более 64 кбайт памяти кажется довольно странным, однако он работает», - не без некоторой иронии замечает по этому поводу Питер Нортон.

Добавим, что, помимо всего прочего, при сегментной адресации один и тот же адрес ОЗУ может быть представлен несколькими комбинациями чисел (в самом деле, одну и ту же сумму можно получить разными способами)

Поскольку современные интелловские процессоры, начиная с 80386, стали 32-разрядными, их регистров хватает, чтобы адресовать до 4 Гбайт памяти. Сегментный способ при этом становится излишним, хотя и сохраняется как вариант ради обеспечения программной совместимости с предыдущими моделями.

Нам осталось рассмотреть еще один очень специфический, но интересный и часто используемый на практике способ адресации данных в ОЗУ. Речь пойдет о работе со стеком. Сам термин происходит от английского слова «stack», имеющего множество значений, причем ни одно из них не подходит достаточно точно. Чаще всего в литературе упоминается магазин револьвера, который по очереди обеспечивает доступ к каждому патрону. Даже если не учитывать ненужную «милитаризацию» терминологии, такой перевод явно не подходит к современной реализации стека. Во-первых, стек не является круговым, а во-вторых, информация в стеке микрокомпьютера не смещается (для точности аналогии пришлось бы признать, что магазин покоится, а вращается револьвер!). Поэтому возьмем термин «стек» как есть, без перевода, и перейдем к определению.

Стек - это неявный способ адресации, при котором информация записывается и считывается только последовательным образом с использованием, так называемого, указателя стека.

Обратимся к примеру. Пусть требуется на время сохранить значения трех целочисленных переменных N1, N2 и N3, а затем их все сразу восстановить. Попробуем воспользоваться для этого стековой памятью. Прежде всего запомним, что стек всегда имеет единственный вход и выход информации - для хранения его адреса и нужен указатель стека. Пусть для определенности он сейчас содержит адрес 2006 (рис. 4.13, а). Тогда по команде «записать в стек N1» (обратите внимание, что в команде не фигурирует явно ни адрес ОЗУ, ни регистр МП!) процессор проделает следующее:

- 1) уменьшит указатель стека на 2 (целое число занимает в памяти 2 байта);
- 2) запишет N1 по полученному адресу 2004 (рис. 4.13, б).

Аналогично при выполнении команд «записать в стек N2» и «записать в стек N3» значения этих переменных попадут в ячейки 2002 и 2000, причем указатель стека станет равным 2000 (рис. 4.13, в).

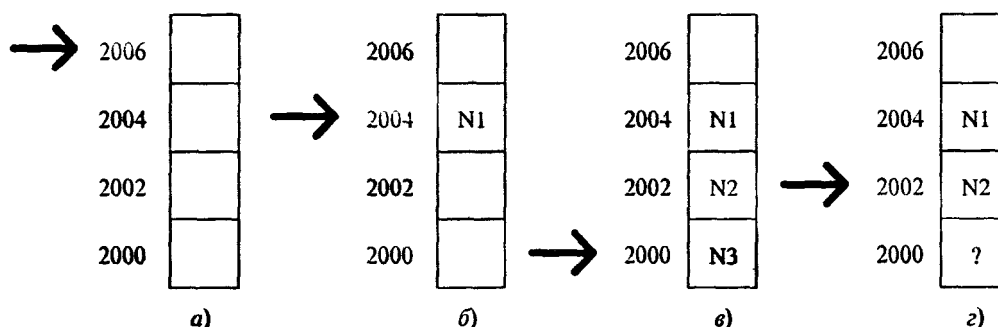


Рис. 4.13. Пример для изучения принципа работы стека

Теперь займемся извлечением информации. Как видно из рис. 4.13, в, указатель сейчас «направлен» на значение переменной N3, а значит считывать придется, начиная с него. Выполним команду «прочитать из стека N3». При этом процессор

- 1) считает в N3 значение из стека;
- 2) увеличит указатель на 2 (рис. 4.13, г).

Аналогично прочитаем N2 и N1, после чего стек опустеет и вернется к начальному состоянию, изображенному на рис. 4.13, а.

Примечание. Обратим внимание на то, что значение в стеке после считывания, конечно же, не исчезает, как это условно показано на рис. 4.13, г, но есть причины полагать, что мы его там можем больше не увидеть. Дело в том, что процессор имеет право иногда временно использовать стек для своих «внутренних» нужд. При этом некоторые ячейки, адреса которых меньше текущего указателя стека, естественно, изменятся. Следовательно, во избежание неприятностей лучше всегда считать, что после считывания информации в стековой памяти она пропадает.

Подводя итоги рассмотрения примера, обратим внимание на следующее. Та информация, которая заносится в стек первой, извлекается последней и наоборот. Обычно об организации стека говорят: «первым пришел - последним ушел». Удобно представить себе аналогию с детской пирамидкой, нижнее кольцо которой невозможно снять до тех пор, пока не будут сняты все остальные.

Еще раз подчеркнем, что в командах работы со стеком адрес ОЗУ не фигурирует в явном виде. Но при этом молчаливо предполагается, что указатель стека уже задан. Как правило, инициализацию указателя стека компьютер делает сам, но не мешает поинтересоваться, так ли это в вашем конкретном случае и достаточно ли памяти под стек имеется при этом. Если указатель стека определен неправильно, то запись в стек может разрушить полезную информацию в ОЗУ и вызвать непредсказуемые последствия.

Интересно, что со стековым способом организации мы имеем дело в жизни гораздо чаще, чем это может показаться на первый взгляд. Во-первых, дисциплине «первым пришел - последним ушел» подчиняются скобки в арифметических выражениях: первой всегда закрывается «самая внутренняя», т.е. как раз последняя скобка. Если вы знакомы с программированием на любом алгоритмическом языке, то без труда найдете и еще два примера: вложенные циклы и подпрограммы.

Итак, мы подробно рассмотрели общие принципы записи информации в стек. Остается сказать, что стековый способ организации ОЗУ используется в вычислительной технике очень широко. На аппаратном уровне процессор «обучен» сохранять в стеке текущий адрес программы при выполнении команды перехода с возвратом, т.е. при вызове подпрограммы. Часто программа предварительно заносит в тот же самый стек необходимые для подпрограммы параметры: так реализуется, например, вызов процедур и функций с параметрами в языке «Паскаль». Кроме того, стек часто используется для временного сохранения значений тех или иных внутренних регистров процессора. Наконец, процессор активно использует стек при реализации прерываний от внешних устройств (об этом будет рассказано в следующем параграфе).

3.4. ФОРМАТЫ ДАННЫХ

Почти каждая команда процессора нацелена на обработку данных, местонахождение которых определяется значениями адресов операндов. Для понимания работы процессора существенно представлять, какого рода данные он может обрабатывать.

Если в ходе исполнения программы ее остановить, прочитать содержимое какой-нибудь ячейки памяти ОЗУ, к которой в это мгновение обращается процессор, и попытаться понять, что именно хранится в этой ячейке, то это чаще всего невозможно сделать, не расшифровав код выполняемой операции - сам по себе хранимый в ячейке двоичный код может быть и числом, и командой, и кодом символа, и чем-то еще. Все дело в том, как его интерпретирует работающая с ним команда.

Перечислим некоторые форматы данных, типичные для 16-разрядной ЭВМ.

8- битовые целые числа без знака. Каждое такое число занимает 1 байт и воспринимается процессором как целое положительное число. Следовательно, диапазон представимости чисел в этом формате от 00000000 до 11111111, т.е. от 0 до FF в шестнадцатеричной системе (от 0 до 255 в

десятичной).

8- битовые целые числа со знаком. В этом случае величина числа задается семью битами, а значение старшего бита определяет знак числа (0 - положительное, 1 -отрицательное). Например, в этом формате код 01101011 означает число +6В.

Однако код 11101011 не означает, как можно подумать, число -6В, так как для кодирования отрицательных чисел применяется, так называемый, дополнительный код. Он образуется следующим образом:

- находится восьмиразрядное двоичное представление абсолютной величины числа;
- найденный код инвертируется, т.е. в нем нули заменяются на единицы и наоборот;
- к полученному коду арифметически прибавляется единица.

Например, процесс получения дополнительного кода десятичного числа -75 таков:

01001011 10110100 10110101

/

Использование дополнительного кода облегчает организацию арифметических действий в процессоре (который, как правило, не имеет даже аппаратно реализованной операции вычитания - все удается свести к сложению целых чисел).

Обратная процедура - восстановление значения числа по дополнительному коду - осуществляется по тому же правилу, что и прямая.

Диапазон представимости чисел в этом формате: от-128 до +127.

16- битовые целые числа со знаком и без знака. Они в точности аналогичны 8-битовым, но код имеет вдвое большую длину. Соответственно, многократно возрастает диапазон представимости: для чисел без знака от 0000 до FFFF (т.е. от 0 до 65535 в десятичной системе), для чисел со знаком - от -8000 до +7FFF (т.е. в десятичной системе от -32768 до +32767).

8- битовые символы. В этом формате двоичный код интерпретируется обрабатывающей его командой как код символа. При работе с персональными ЭВМ обычно используется система кодирования ASCII, о которой говорилось в главе 1. В этой системе стандартизированы (закреплены за определенными символами) коды, у которых значение старшего бита равно 0; все прочие коды остаются за символами национальных алфавитов и дополнительными специальными символами.

Битовые поля. В этом формате значащим является не весь 8- или 16- разрядный код в целом, а каждый из составляющих его битов. Один из примеров битового поля - содержимое регистра состояния процессора (см. выше). Другой пример -форма хранения множеств в языке «Паскаль».

Существуют и другие форматы данных - двоично-десятичные числа, строки и т.д.

3.5. ОБРАБОТКА ПРЕРЫВАНИЙ

Важную роль в работе современного МП играют прерывания. Они всегда нарушают естественный ход выполнения программы для осуществления неотложных действий, связанных, например, с реакцией на щелчок мыши или сбой в цепи электропитания.

События, вызывающие прерывания, можно разделить на две группы: *фатальные* и *нефатальные*. На фатальные (неотвратимо наступающие) процессор может реагировать единственным способом: прекратить исполнение программы, проанализировать событие и принять соответствующие меры (чаще всего - сообщить причину прерывания пользователю и ждать его реакции). Однако часто можно с остановкой программы повременить: запомнить, что прерывание было, и продолжать исполнять программу. Например, сложение с переполнением разрядной сетки -фатальное событие, после которого остановка неизбежна; попытка вывода на принтер, не готовый к приему информации, может быть отложена (с сохранением этой информации).

Основные виды прерываний - *внутрипроцессорные прерывания* и *прерывания от внешних устройств*. Первые связаны с возникновением непреодолимого препятствия при выполнении программы. Причин может быть много: из памяти выбрана команда с несуществующим кодом или адресом, в ходе исполнения команды возникло переполнение разрядной сетки ЭВМ или произошла попытка записи в оперативную память, отведенную другой задаче. В большинстве подобных случаев дальнейшее выполнение программы становится невозможным и управление передается системе, обеспечивающей прохождение задач (чаще всего это - операционная система), которая и

принимает меры по обработке внешней нештатной ситуации.

Гораздо реже, но возможны, нефатальные внутрипроцессорные прерывания, например, специальные отладочные прерывания. Такие прерывания обеспечивают возможность пошагового исполнения тестируемой программы под контролем специальных программных средств.

Прерывания второй группы возникают по требованиям какого-либо из многочисленных внешних устройств. Такое событие обычно не приводит к фатальному завершению задачи, напротив, это вполне нормальная ситуация. Поэтому при таком прерывании МП принимает меры для того, чтобы обеспечить дальнейшее выполнение программы: запоминает в стеке текущее значение счетчика команд и содержимое регистра состояния. Затем происходит переход на подпрограмму обработки данного прерывания. После ее выполнения процессор восстанавливает из стека значения своих регистров и продолжает выполнение прерванной программы.

Прерывания от внешних устройств подробно рассматриваются в следующем пункте.

Отметим, что термин «прерывание» часто используется еще в одном значении. Речь идет о, так называемых, программных прерываниях. Например, для IBM-совместимых компьютеров существуют многочисленные команды прерывания 1NT с самыми разнообразными номерами. Следует понимать, что 1NT - это одна из инструкций процессора; чтобы она заработала, ее код должен содержаться в программе. В противоположность этому, «настоящие» прерывания возникают аппаратно и не требуют наличия каких-то специальных команд в тексте прерываемой программы. Более того, аппаратное прерывание может произойти между двумя любыми командами программы.

К программным относятся и межмашинные прерывания, возникающие в локальной сети при обмене информацией между компьютерами.

Запрет нефатального прерывания называется *маскировкой*; маскировка задается программистом или системной программой. Делается это либо с помощью установления вида битового поля в специальном регистре маски прерываний, в котором значения разрядов (0 или 1) связаны соответственно с отсутствием или наличием маскировки закрепленного за этим разрядом прерывания, либо с аналогичным использованием разрядов регистра состояния процессора.

После получения сигнала о незамаскированном прерывании процессор делает следующее:

- запоминает состояние прерванной программы;
- распознает источник прерывания;
- вызывает и выполняет специальную системную программу обработки прерываний;
- восстанавливает состояние прерванной программы и, при возможности, продолжает ее исполнение.

Для распознавания источника прерывания анализируются некоторые биты регистра состояния процессора, состояние внешних устройств и т.д. Для запоминания состояния прерванной программы чаще всего используется стек. Выше мы уже говорили о стеке в ракурсе организации особого способа адресации. Кроме того, стек используется процессором как для организации механизма прерываний, так и для обработки обращения к подпрограммам, передачи параметров и временного хранения данных.

Назначение программы обработки - понять и в удобной для пользователя форме вывести (обычно на экран) сообщение о причине прерывания и (иногда) дать рекомендации по возможной реакции на эту причину.

Все это разительно отличается от ситуации с машинами 1-го и 2-го (а отчасти и 3-го) поколений, когда пользователь без помощи системного программиста в большинстве случаев не мог разобраться в причине события, вызвавшего прерывание его программы на этапе исполнения. Такая ситуация стала нетерпимой для пользователей персональных компьютеров, и чем совершеннее анализатор программы обработки прерываний, тем выше уровень «дружелюбности» пользовательского интерфейса.

3.6. РАБОТА МИКРОПРОЦЕССОРА С ВНЕШНИМИ УСТРОЙСТВАМИ

Выше было описано, как процессор обменивается информацией с ее наиболее важным и оперативным источником - памятью. Рассмотрим теперь, как МП может принять данные или передать их внешним устройствам. Способ решения этой задачи в зависимости от конструкции ЭВМ (подчеркнем: не от конструкции МП, а от конструкции всей ЭВМ!) может быть одним из двух

указанных ниже:

- 1) устройства ввода-вывода включаются в общее адресное пространство;
- 2) устройства ввода-вывода имеют собственное адресное пространство.

В первом случае при обращении к определенным адресам памяти вместо обмена с ОЗУ происходит аппаратное подключение того или иного внешнего устройства. При этом для «общения» с внешними устройствами и с памятью используются одни и те же команды МП, хотя, конечно, обмен с внешним устройством протекает по более сложному протоколу, чем с памятью.

Во втором случае внешние устройства образуют отдельное адресное пространство, обычно значительно меньшее, чем у ОЗУ. Каждая ячейка этого дополнительного адресного пространства называется *портом*. Каждому внешнему устройству обычно соответствует несколько портов с последовательными адресами. Обмен процессора с организованными подобным образом устройствами осуществляется специальными командами ввода-вывода.

На самом деле, с точки зрения схемной организации оба описанных способа имеют очень много общего. При любом обмене, будь то обращение к ОЗУ, ПЗУ или внешнему устройству, процессор выставляет адрес информации на единую адресную шину, а данные передает или принимает по общей шине данных. Выбор же требуемого адресата - ячейки памяти или порта - осуществляется подачей специального управляющего сигнала.

В одном и том же компьютере могут встречаться оба способа адресации устройств ввода-вывода одновременно. Так, в ПЭВМ «Ямаха» накопитель на магнитных дисках включен в общее адресное пространство памяти, а печатающее устройство оформлено в виде нескольких портов. Некоторые порты этого компьютера служат для подключения к небольшому 64-килобайтному адресному пространству 128 кбайт ОЗУ и многочисленных ПЗУ. При этом все ресурсы памяти разбиваются на отдельные страницы по 16 кбайт каждая, и «активными» в каждый момент времени могут быть только четыре из них.

Рассказ о работе с периферийными устройствами был бы неполным без описания того диалога, который ведет с ними МП. Рассмотрим этот диалог на примере простейших устройств ввода-вывода - например, печатающего устройства.

Обмен с подключенным к ЭВМ печатающим устройством производится через два основных порта - порт состояния и порт данных. В первом хранится информация о состоянии устройства в данный момент времени, а во второй МП помещает данные для вывода на бумагу. Каждый бит порча состояния хранит ответ на вполне определенный вопрос: заправлена ли бумага, готов ли принтер принять данные от компьютера и т.д. Все это для МП - входные сигналы. Но есть и выходные, которые через порт передаются от МП к печатающему устройству. Наиболее важным из них является бит, свидетельствующий о готовности информации в порту данных к передаче. Этот управляющий сигнал часто называют *стробом*.

В наиболее простом случае обмен информацией между процессором и принтером может протекать следующим образом. Пусть МП должен вывести на печать какой-нибудь символ. Он считывает порт состояния принтера и анализирует содержимое его бита готовности. Если результат положительный, т.е. печатающее устройство готово принять информацию, обмен продолжается, в противном случае МП снова считывает порт состояния и повторяет анализ. Когда процессор получит от принтера сигнал о готовности к обмену, он заносит требуемый символ в порт данных и установкой стробирующего сигнала сообщает об этом принтеру. Затем МП снова периодически считывает порт состояния, но следит уже за другим битом (через этот бит принтер сообщит процессору о том, что данные приняты, т.е. скопированы в собственное ОЗУ принтера). После этого МП убирает стробирующий потенциал и продолжает работу по программе.

Конечно, реальный диалог современного компьютера с печатающим устройством сложнее. Вот пример. Вы, наверное, заметили, что описанный выше алгоритм «зациклится», если принтер не подключен или не готов к работе: процессор будет ждать сигнала о готовности принтера, а того не будет. Выход - ожидать сигнала готовности в течение какого-то времени, а затем выдать пользователю сообщение о неготовности принтера к работе. При этом желательно проверить содержимое других битов порта состояния (например, отвечающих за наличие бумаги и т.п.) и уточнить, если потребуется, диагностическое сообщение.

Обмен информацией между МП и устройством ввода следует рассмотреть отдельно, так как он может иметь некоторые существенные особенности. Описанная выше идеология передачи данных, конечно, может быть использована и для устройств ввода - например, для клавиатуры. В

этом случае программа в тот момент, когда ей потребуются входные данные, опросит состояние клавиатуры и примет с нее данные. Однако для сложных современных программных систем такой метод неудобен из-за существенного замедления времени реакции ЭВМ. В самом деле, если некоторая сложная программа занята вычислениями, а вы хотите ее прервать, то придется подождать, пока процессор освободится и займется опросом клавиатуры (собственно говоря, тогда и прерывать-то будет нечего!). Все это сильно напоминает очередь в приемной бюрократа-начальника, когда люди вынуждены часами ждать, пока их вызовут для пятиминутного решения вопроса.

Для предотвращения подобных неприятностей во всех ЭВМ на базе МП наряду с программным опросом устройств ввода существует еще один механизм - механизм *прерывания от внешних устройств*. Такие прерывания настолько широко используются в компьютерах, что имеет смысл рассмотреть их подробнее. Для понимания сути работы прерываний снова обратимся к аналогии с руководителем. Пусть в его кабинете идет совещание по подведению итогов деятельности предприятия, и в этот момент по телефону поступает очень важная информация, требующая немедленного принятия решения. Как в таком случае обычно развиваются события? Секретарь, не дожидаясь конца совещания, сообщает шефу о звонке. Тот, прервав свое выступление, снимает трубку и выясняет суть дела. Затем он либо тут же принимает решение и сообщает его, либо предлагает пока сделать самое необходимое, а после совещания обещает перезвонить и дать дальнейшие указания. Затем (обратим внимание на эту деталь!) он произносит что-нибудь в духе «Так, на чем мы остановились?» и продолжает совещание как ни в чем не бывало. Впрочем, если ситуация экстренная, то совещание может быть прекращено или в его ход могут быть внесены определенные коррективы.

Вернемся к компьютеру. Каждое его устройство (клавиатура, мышь, дисковод и др.) способно затребовать внимание процессора, выставляя сигнал требования прерывания. Процессор проверяет наличие этого сигнала после выполнения каждой операции. «Увидев» требование прерывания, МП немедленно начинает его обрабатывать. Прежде всего, он запоминает свое текущее состояние (счетчик команд и регистр состояния) с тем, чтобы в дальнейшем иметь возможность продолжить выполнение прерванной программы. После этого происходит переход на программу, обрабатывающую данное прерывание. Заметим, что все перечисленные действия реализуются на аппаратном уровне, т.е. фактически являются «врожденными рефлексам» МП.

Программа обработки прерывания, получив управление, проводит оперативный анализ причины прерывания. Если причина серьезная и требует немедленных действий, то эти действия выполняются. Если же с обработкой можно подождать (например, просто нажата обычная символьная клавиша), в специальную область памяти (буфер) заносится информация о происшедшем прерывании. Во всех случаях прерывания завершаются выполнением специальной команды возврата, которая восстанавливает содержимое программного счетчика и регистра состояния, давая тем самым микропроцессору возможность продолжить работу прерванной программы. Естественно, что программа обработки прерывания должна была предварительно восстановить все испорченные ей рабочие регистры МП.

Реальная картина, конечно, еще сложнее: прерывания от разных устройств имеют разные уровни приоритета, существует возможность маскировки прерываний и т.п. Однако сути дела все это существенно не меняет.

В последнее время необходимость понимания механизма работы прерываний сильно возросла в связи с возникновением идеологии программирования по событиям. Она связана с распространением среды «Windows» и лежит в основе систем типа «Visual Basic» или «Delphi». Приведем примеры нескольких событий, на которые программа может реагировать: сдвинута мышь, нажата (или отпущена) клавиша мыши, нажата клавиша «ввод», выбран тот или иной пункт меню, открыто новое окно на экране и многие-многие другие. Полный перечень событий занимает в описании несколько страниц. Интересно, что программа на таком языке уже не является единым целым: на каждое событие пишется своя собственная программа, хотя все они и могут быть связаны между собой.

Итак, мы рассмотрели наиболее общие принципы работы микропроцессоров вне зависимости от их модели. Теперь познакомимся с конкретными процессорами.

3.7. ПРИМЕР: СИСТЕМА КОМАНД ПРОЦЕССОРОВ СЕМЕЙСТВА PDP

В качестве примера реально существующего процессора, удобного для более детального изучения, возьмем процессор серии машин, созданных фирмой DEC (США) и известных под названием PDP-11 (в нашей стране аналогичные процессоры использовались в семействах 16-разрядных мини- и микро-ЭВМ «Электроника», ДБК, БК, а также в школьном компьютерном классе УКНЦ; всего таких машин было выпущено в СССР до 1,3 млн. штук и примерно столько же в США). Это семейство мини- и микро-компьютеров - одно из самых долгоживущих: первая машина этой серии была выпущена в 1970 г., а прекращение выпуска было запланировано фирмой DEC на 1997 г.

Причина выбора процессора PDP для иллюстрации обсуждаемого вопроса состоит в том, что система команд этого процессора построена на простых и наглядных принципах, изучив и запомнив которые уже можно составлять несложные программы. В то же время система команд других процессоров, например, широко распространенных представителей семейства «Intel», устроена значительно сложнее, требует запоминания большого количества справочных данных. В качестве подтверждения сказанного достаточно указать, что в команде процессора PDP может быть использован любой из имеющихся внутренних регистров, тогда как многие команды процессоров «Intel» оперируют с фиксированными регистрами, не допуская альтернативного расположения данных и результатов.

Процессор машин серии PDP и его отечественные аналоги (при дальнейшем изложении будем все это обширное семейство для краткости называть «процессор PDP») с точки зрения программиста устроен довольно просто. Он состоит из восьми *регистров общего назначения* (РОН) и *особого регистра*, в котором отображается текущее состояние процессора (в иностранной литературе его принято обозначать PSW - Processor Status Word). Любой из регистров общего назначения может быть использован в командах на равных основаниях. Вместе с тем имеются два выделенных регистра, содержимое которых процессор использует для собственных нужд. Прежде всего следует назвать регистр R7, выполняющий роль счетчика команд, в котором хранится адрес следующей инструкции программы. Другим выделенным регистром является указатель стека R6, используемый при запоминании информации в момент вызова подпрограмм и при переходе к обработке прерывания.

Остальные 6 регистров, обозначаемые R0, R1, R2, R3, R4, R5, программист может полностью использовать по своему усмотрению. Имеется только одно (непринципиальное) исключение: в обширной системе команд PDP существует лишь единственная, и то достаточно экзотическая, команда (MARK), работающая с регистром R5.

Следует отметить, что и выделенные регистры процессора R6 и R7 могут быть использованы в любой команде наряду с обычными РОН R0-R5. Например, допускается переписать информацию из R7 в R3 и тем самым сообщить программе адрес памяти, в котором находится следующая команда программы. Такой прием очень часто используется программистами для автоматической «самонастройки» на те адреса ОЗУ, в которых оказалась загруженной программа. Интересно, что такого простого доступа к содержимому программного счетчика многие процессоры не имеют.

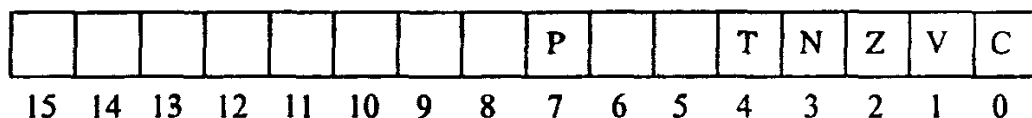


Рис. 4.14. Схема регистра состояния процессора PDP

Регистр состояния процессора PSW, как и все РОН, является 16-битным. Его особенностью является то, что каждый его бит имеет самостоятельное значение и может использоваться процессором отдельно от других. В управляющих битах регистра постоянно отображается информация о выполняемых результатах (отрицательность, равенство нулю и т.п.), а также о состоянии процессора в данный момент (например, один из битов позволяет разрешать или запрещать обработку прерываний). Из всех используемых битов (не все 16 бит регистра состояния задействованы!) наиболее используются два; их принято обозначать N и Z. Управляющий признак N (Negative) отражает знак результата операций: если число отрицательное, то N = 1, неотрицательное - N = 0. Бит Z (Zero) говорит о равенстве или неравенстве нулю результата: в первом случае Z = 1, во втором - Z = 0. Путем несложных рассуждений можно убедиться в том, что из шести известных мате-

$$=, <, >, \leq, \geq, \neq$$

четыре могут быть проанализированы по одному из признаков, а два оставшихся требуют совместного анализа битов N и Z.

Другие управляющие биты, изображенные на рисунке, используются следующим образом. Бит P - задание маскировки прерываний от внешних устройств (P = 0 - прерывание состоится, P = 1 - замаскировано). Бит T - «признак ловушки»; при T = 1 после прерывания запускается специальная системная программа, позволяющая на этапе отладки текущей программы осуществлять режим трассировки. Бит V - признак переполнения разрядной сетки при арифметической операции (если это произошло, устанавливается V = 1); бит C - аналогично при переполнении разрядной сетки при логической операции, сопровождаемой сдвигом кода. Описанные управляющие признаки широко используются для реализации разветвлений программы в зависимости от полученных результатов, анализа причин прерываний и других действий.

Процессор PDP имеет достаточно удобный и широкий набор команд. Основная их часть является двухадресными и одноадресными, т.е. они обрабатывают два или один операнд, соответственно. Для выполнения некоторых управляющих команд данные не требуются (например, команда остановки программы) - такие команды не содержат ссылок на операнды. Форматы одно- и двухадресных команд изображены ниже на рис. 4.15.

КОП									КМА				N ПОН		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

КОП			КМА1			N ПОН1			КМА2			N ПОН2			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Рис. 4.15. Формат одноадресной команды (вверху) и двухадресной (внизу). КОП - код операции.

КМА - код метода адресации, N РОН - номер регистра общего назначения (код адреса)

Примеры некоторых наиболее важных операций приведены в табл. 4.2. Действия, выполняемые по простейшим арифметическим операциям, понятны из пояснений. В некоторых дополнительных пояснениях нуждаются только команды переходов.

Во-первых, *переходы* бывают *абсолютные* (на заданный адрес) и *относительные* (на определенное число команд относительно данной). Переходы первого типа более наглядны и могут быть реализованы на любой адрес ОЗУ. Относительные переходы требуют вычисления адреса перехода, зато они не привязаны к конкретным адресам ОЗУ: программа, использующая относительные переходы, может работать в любом месте памяти. Относительные переходы занимают меньше места в памяти, чем абсолютные; вследствие этого диапазон относительных переходов ограничен (в процессоре PDP только на 127 слов вперед и 128 - назад).

Таблица 4.2
Наиболее важные команды процессора PDP

Мнемоника команды	Содержание команды	Пояснения
HLT	Остановка	Выполнение программы прекращается
CLR A	Очистить A	A:=0 (в операнд A засылается 0)
INC A	Увеличить A на 1	A:=A+1 (значение A увеличивается на 1)
DEC A	Уменьшить A на 1	A:=A-1 (значение A уменьшается на 1)
MOV A,B	Переписать A в B	B:=A (B принимает значение A)
ADD A,B	Сложить A в B	B:=A + B (сумма помещается в B)
SUB A,B	Вычесть A из B	B:=B-A (разность помещается в B)
CMP A,B	Сравнить A с B	По знаку разности A-B устанавливаются управляющие биты; A и B сохраняются

JMP A	Безусловный переход к A	Безусловный переход по адресу A
BR K	Безусловный переход на K.	Безусловный переход через K команд
BEQ K	Переход по $= 0$	Переход на K слов, если результат $= 0$
BNE K	Переход по $\neq 0$	Переход на K слов, если результат $\neq 0$
BPL K	Переход по ≥ 0	Переход на K слов, если результат ≥ 0
BMI K	Переход по < 0	Переход на K. слов, если результат < 0
JSR A	Вызов подпрограммы по адресу A	Переход на адрес A с запоминанием адреса возврата для команды RET
RET	Возврат из подпрограммы	Возврат на команду, следующую за вызовом

Во-вторых, переходы делятся на *безусловные* и *условные*. Безусловные, как. следует из названия, происходят всегда. При выполнении условного перехода анализируются управляющие биты. Переход происходит только в том случае, когда признаки имеют требуемые значения; в противном случае переход игнорируется и выполняется следующая за ним команда. Чаще всего условному переходу предшествует команда сравнения, «подготавливающая» управляющие признаки для анализа. Отметим, что команды BEQ и BNE проверяют значение признака Z, а BPL и BMI - признака N.

В таблице у команд перехода для наглядности указаны адреса или величина смещения в словах. Именно таким образом и кодируются команды в памяти ЭВМ. Однако при записи текстов программ дело обычно обстоит несколько проще: в требуемых местах программы ставятся метки, а расчет конкретных адресов и смещений машина производит самостоятельно (см. приводимые ниже примеры).

В приведенных в таблице командах в качестве операндов фигурируют условные обозначения A и B. Расшифруем их. Система команд процессора PDP построена достаточно логично и закономерно, поэтому любой операнд- и первый, и второй - в любой операции задается совершенно одинаково. В PDP существует 8 различных способов задать местонахождение информации, требующейся для выполнения операции. Их принято называть *методами адресации*, и все они связаны с различным использованием РОН. Наличие развитой системы методов адресации делает программирование для процессора PDP очень гибким и мощным.

Опишем коротко организацию основной памяти, контролируемой обсуждаемым микропроцессором. Основная адресация - байтовая; каждый байт имеет уникальный адрес. Так как процессор является 16-разрядным, то объем контролируемого им номинально адресного пространства 64 кбайта; на деле программисту для размещения программы и данных доступно меньшее пространство, так как 8 кбайт памяти зарезервировано для общения с внешними устройствами. Реально, однако, с помощью специальных механизмов преобразования исполнительных адресов емкость ОЗУ, доступную пользователю, делают значительно большей (детали рассматривать не будем).

Байт может быть операндом для некоторых команд. Однако, большая часть команд обрабатывает операнды длиной 2 байта - «слово». Слово состоит из двух соседних байтов; адресом слова считается адрес младшего входящего в него байта.

Таблица 4.3
Некоторые способы указания операндов в командах

Название	Мнемоника	Пояснение
Регистровая	Rn	Операнд в регистре Rn
Косвенно-регистровая	(Rn)	Операнд в ячейке ОЗУ, адрес которой в Rn
Автоинкрементная	(Rn)+	То же, но после чтения Rn увеличивается на 1
Автодекрементная	-(Rn)	То же, но перед чтением Rn уменьшается на 1

В качестве иллюстрации разберем выполнение команды MOV(R1)+,RO

По этой команде сначала считывается информация из памяти по адресу, находящемуся в R1. После чтения значение R1 автоматически увеличивается, так что при последующем выполне-

нии данной команды будет обрабатываться следующая ячейка. Операция завершается записью считанной из ОЗУ информации в регистр R0.

Из разобранного примера отчетливо видно, что имеющиеся у процессора PDP способы адресации позволяют удобно работать не только с одиночными данными, но и с массивами последовательно хранящейся в памяти информации. Кроме того, некоторые приемы использования счетчика команд R7 в качестве РОН могут давать еще более интересные эффекты. Пусть, например, в некоторой программе встречается команда

MOV (R7) + , R2

и в следующем за ней слове хранится число 6. Учитывая, что после извлечения команды из ОЗУ счетчик R7 немедленно увеличивается, в момент выполнения команды его содержимое уже показывает на слово памяти, где хранится число 6. Следовательно, в качестве операнда (R7) будет считано именно оно, и, кроме того, R7 автоматически «передвинется» на следующую за константой ячейку благодаря автоинкрементному способу адресации. Таким образом, описанный прием позволяет использовать в качестве операнда константу, хранящуюся непосредственно в команде. Завершая обсуждения основных принципов программирования процессора PDP, приведем примеры линейного, разветвляющегося и циклического фрагмента программы.

Пример 1. Программа вычисления по формулам $R1:=R2+R3$; $R4:=R3-R2$.

Программа 131

```
MOV R2, R1          ; сразу складывать нельзя, так как сумма заменяет
                    ; второй операнд
ADD R3, R1
MOV R3, R4
SUB R2, R4
HLT
```

Пример 2. В R1 и R2 хранятся некоторые числа. Поместить большее из них в R5, а меньшее - в R0.

Программа 132

```
CMP R1, R2          ; сравнить R1 и R2
BPL L1             ; переход при  $R1 \geq R2$ , если  $R2 > R1$ 
MOV R1, R0
MOV R2, R5
BR L2
; если  $R1 > R2$ 
LI: MOV R1, R5
MOV R2, R0
L2: HLT
```

Примечание. В тексте программы указаны переходы на метки, но в результате трансляции они будут автоматически преобразованы в переходы через заданное число слов.

Пример 3. Пусть в R1 задан начальный адрес массива ОЗУ, а в R2 - число ячеек в нем. Вычислить сумму элементов массива.

Программа 133

```
CLR R0              ; очистка суммы
LI: ADD (R1) + , R0  ; добавить к сумме очередной элемент
DEC R2              ; уменьшить на единицу число оставшихся элементов
BNE L1              ; цикл, пока не останется 0 элементов
HLT
```

Примечание. В программе используется тот факт, что команда DEC автоматически сравни-

вает результат с нулем, что делает ненужной специальную команду сравнения СМР.

Контрольные вопросы и задания

1. Назовите преимущества, которые имеют интегральные микросхемы перед дискретными элементами. Ответ постарайтесь обосновать.
2. Для каких функций первоначально предполагалось использовать микропроцессор?
3. Что такое разрядность МП? Чем она определялась и как изменялась? Какую максимальную разрядность имеют современные процессоры?
4. Вспомните, что вы читали о новых моделях микропроцессоров.
5. Какие микропроцессоры вы знаете? Известен ли вам тип процессора в компьютерах, с которыми вы имеете дело?
6. Подумайте, в каких устройствах, с которыми вы сталкиваетесь в повседневной жизни, могут быть микропроцессоры,
7. Что представляет собой микропроцессор с точки зрения программиста?
8. Каковы функции микропроцессора в целом?
9. Какие функции могут выполнять регистры процессора?
10. Вспомните основной цикл работы процессора ЭВМ. Какую роль в нем играет счетчик адреса команд?
11. От чего зависит объем контролируемого 32-разрядной ЭВМ адресного пространства?
12. Что такое методы адресации? Какие методы адресации ОЗУ вы знаете?
13. Объясните, как работает метод косвенной адресации.
14. Приведите 3-4 варианта сегментных адресов, указывающих на одну и ту же ячейку памяти.
15. Охарактеризуйте наиболее употребимые форматы данных.
16. Каким образом процессор может адресоваться к устройствам ввода-вывода? Что такое порт ввода-вывода?
17. Опишите примерный ход взаимодействия между процессором и принтером. Как вы считаете, что происходит раньше: очередной символ печатается на бумагу или появляется сигнал готовности к приему следующего?
18. Что такое прерывание и как оно работает? Проанализируйте приведенные в тексте аналогии и попробуйте придумать собственные.
19. Имеют ли место прерывания в следующей ситуации и почему: во время активной работы программы пользователь нажимает клавишу мыши и окно на экране, соответствующее другой программе, при этом сворачивается?
20. Каковы функции регистра слова состояния процессора?
21. Какие способы адресации используются в процессоре PDP?
22. В чем особенности реализации условных и безусловных переходов в процессоре PDP?
23. Как организована работа с операндами?

§4. УЧЕБНАЯ МОДЕЛЬ МИКРОКОМПЬЮТЕРА

В предыдущем разделе были кратко изложены наиболее важные черты реально существующего простого процессора ЭВМ. Конечно, на самом деле все устроено заметно сложнее, поэтому мы ограничились достаточно поверхностным знакомством. Более современные процессоры устроены еще сложнее.

Существует альтернативный подход - изучение сложного объекта на упрощенной модели, сохраняющей все его наиболее существенные черты. Именно такой подход использован в данном разделе. Рассмотрение фундаментального материала, освобожденного от второстепенных технических деталей, помимо простоты имеет и еще одно преимущество - полученные знания не так быстро «стареют».

Прием, состоящий в объяснении принципов работы ЭВМ на базе учебной модели, возник довольно давно. Имеется целый ряд учебных моделей: «Кроха», «Нейман», «Малютка» и другие (описание которых можно найти в рекомендуемой в конце главы дополнительной литературе).

В данном параграфе рассматривается учебная модель микропроцессорной ЭВМ четвертого поколения, предложенная преподавателем Пермского педагогического университета

Е.А.Ереминым. Модель носит название «Е97» В нее включены наиболее характерные черты устройства современных компьютеров. Модель не повторяет полностью ни один реальный компьютер, хотя дает достаточно полное представление об его устройстве. В «Е97» нет излишних технических деталей, он достаточно прост и содержит только то, что необходимо для изучения.

«Е97» может работать как с числовыми, так и с нечисловыми данными, отражает общепринятую байтовую организацию памяти, способен обрабатывать массивы информации (поскольку имеет современные методы адресации), реалистически отражает работу с периферийными устройствами современного компьютера. Существенно, что наряду с теоретической моделью компьютера создана программа-эмулятор, позволяющая имитировать «Е97» на различных компьютерах, в том числе IBM-совместимых. Ряд непринципиальных ограничений и нереализованных возможностей, указанных ниже, относятся именно к этой программе-эмулятору.

4.1. СТРУКТУРА УЧЕБНОГО МИКРОКОМПЬЮТЕРА

В состав учебного микрокомпьютера входят следующие устройства (рис. 4.16):

центральный процессор, память двух видов (ОЗУ и ПЗУ), а также два наиболее важных внешних устройства (клавиатура для ввода информации и дисплей для вывода; оба устройства, как принято в современных ЭВМ, подключены через обеспечивающие согласование контроллеры).

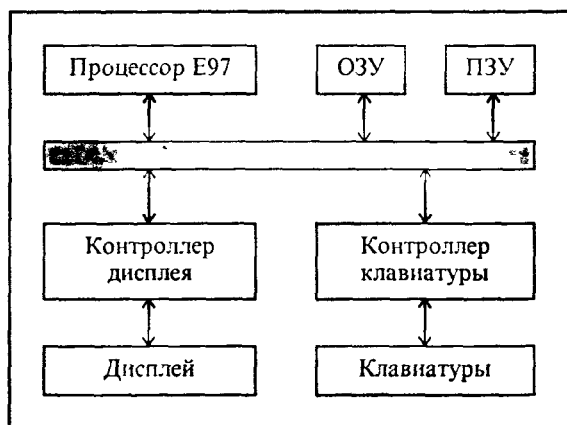
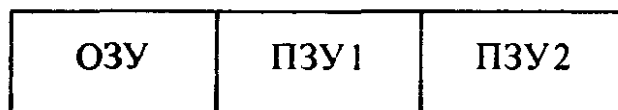


Рис. 4.16. Схема устройства учебного компьютера «Е97»

Главным блоком компьютера служит 16-разрядный процессор «Е97», способный работать как с двухбайтовыми словами, так и с отдельными байтами. Таким образом, он может оперировать с данными разной длины. Познакомившись с тем, как «Е97» обрабатывает разные типы информации, читатель легко сможет в будущем обобщить логику «один байт или много» на случай большей разрядности процессора.

В процессоре имеются внутренние регистры памяти, при помощи которых реализован метод косвенной адресации к ОЗУ. Очевидно, что полное 16-разрядное адресное пространство «Е97» позволяет напрямую адресовать до 64 кбайт памяти; для учебной ЭВМ это более чем достаточно. Поэтому реально существующей памяти будут соответствовать лишь некоторые диапазоны адресов (рис. 4.17). Для первой программной реализации модели «Е97» приняты следующие значения констант: LoROM = 4000, HiROM = 4180.



0000 LoROM HiROM FFFF

Рис. 4.17. Структура памяти микрокомпьютера «Е97» (LoROM - нижняя граница ПЗУ 1. HiROM-верхняя граница ПЗУ 1, совпадающая с началом ПЗУ 2)

В «Е97», как и в реальном компьютере, существует память двух видов -оперативная (ОЗУ) и постоянная (ПЗУ) В первой хранится текущая информация (т.е. программа и данные) по решаемой задаче, причем она может как считываться, так и записываться. Во второй, предназначенной

только для считывания, содержатся разработанные при проектировании ЭВМ подпрограммы наиболее важных и часто используемых действий, среди которых важное место занимают алгоритмы обмена с внешними устройствами.

Видеопамять в «Е97» размещается в контроллере дисплея; для простоты модели будем считать, что видеопамять не входит в адресное пространство процессора. Доступ к видео-ОЗУ осуществляется путем обращения к внешнему устройству.

Как уже отмечалось, минимальной адресуемой ячейкой памяти в современных ЭВМ является байт. Все байты в памяти «Е97» пронумерованы и их 16-разрядные номера находятся в пределах от 0000 до FFFF. Реальные адреса должны попадать в выделенные на рис. 4.17 области. При обращении к любому другому адресу происходит аварийное прекращение выполнения программы.

Байты в памяти могут объединяться в слова, которые для данного компьютера состоят из двух соседних байтов (у современных процессоров обычно из четырех, но для понимания основных принципов это не очень существенно). По традиции примем, что слово адресуется наименьшим из номеров байтов, причем этот адрес соответствует младшему байту слова. Для удобства изображения содержимого ОЗУ на экране дисплея будем считать, что адрес слова *всегда является четным*

Для наглядности проиллюстрируем все эти важные положения примером (рис. 4.18). Слово с адресом 40 имеет значение FFOO, а слово с адресом 42 - 1234; адреса 41 и 43 для слов запрещены. Обратите внимание на то, что байт 40 имеет нулевое значение, а байт с адресом 41 - значение FF. Иначе говоря, байты слова хранятся в памяти «задом наперед» - *сначала младший, а затем старший*. Именно так хранится слово в большинстве компьютеров.

Рассмотрим теперь роль ПЗУ. Его наличие дает целый ряд преимуществ, важнейшими из которых являются следующие.

40	41	42	43
00	FF	34	12

Представление в байтах

41	FF	00	40
43	12	34	42

Представление в словах

Рис. 4.18. К вопросу о представлении информации в «Е97»

Во-первых, в современных ЭВМ не всякое элементарное действие реализуется в виде команды процессора - многие из них выполняются по программе. В «Е97», например, по программе происходит нахождение остатка от деления, вычисление модуля, перевод целого числа из двоичной системы в десятичную. Наличие таких подпрограмм освобождает модель от необходимости вводить в систему команд учебного процессора разные нестандартные и несуществующие в реальных процессорах команды типа «вывод числа на экран в десятичной форме» и ей подобные.

Во-вторых, наличие ПЗУ позволяет не конкретизировать алгоритм работы с внешними устройствами на уровне системы команд процессора, а перенести их в ПЗУ (именно так делается и в настоящих компьютерах). ПЗУ такого рода носит специальное название **BIOS** (от английского Basic Input Output System - **базовая система ввода-вывода**) и обслуживает конкретные типы внешних устройств, входящих в состав машины BIOS имеет стандартные точки входа, к которым и обращается все программное обеспечение при необходимости произвести обмен информации с внешним устройством.

Завершая обсуждение ПЗУ, отметим еще одну деталь. В данной учебной ЭВМ оно условно делится на две части, названные на рис. 4.17 ПЗУ1 и ПЗУ2. ПЗУ1 доступно для изучения и представляет собой настоящие программы в кодах процессора «Е97». Другая часть долговременной памяти - ПЗУ2 - напротив, «скрыта» от просмотра. Определены лишь входные точки и их назначение; в основном это операции ввода. Деление на ПЗУ1 и ПЗУ2 является чисто техническим приемом и не имеет никаких аналогий в реальных ЭВМ (наличие «закрытого» ПЗУ2 просто позволяет легче и быстрее осуществить программный имитатор учебного микрокомпьютера).

Перейдем к рассмотрению контроллеров внешних устройств В учебном микрокомпьютере они фигурируют в виде портов ввода-вывода. Каждому из устройств соответствует два таких порта: **регистр данных и регистр состояния**. Через первый происходит обмен информацией, а вто-

рой позволяет этот обмен синхронизировать. Если в знаковый (старший) бит младшего байта регистра состояния записана единица, то это означает, что устройство к обмена готово.

Отметим, что контроллеры современных периферийных устройств имеют обычно более двух портов. Например, в IBM-совместимых компьютерах для работы со стандартным печатающим устройством используется три порта - к уже названным добавляется регистр управления. Ничего принципиально нового наличие дополнительных портов не вносит.

Такова внутренняя структура учебного микрокомпьютера. Рассмотрим теперь устройство самого процессора, каким он видится программисту. «Е97» состоит из семи 16-разрядных регистров. 4 регистра общего назначения **RO - R3**, счетчика адреса команд **PC** (Program Counter), указателя стека **SP** (Stack Pointer) и регистра состояния процессора **PS** (Processor Status), в котором мы будем использовать только два младших бита **N** и **Z**. «Поведение» этих управляющих битов согласно общепринятым закономерностям следующее:

$N = 0$ - результат ≥ 0 , $N = 1$ - результат < 0 ,
 $Z = 0$ - результат $\neq 0$, $Z = 1$ - результат $= 0$.

Здесь «результат» означает результат последней из выполненных арифметических операций. Все эти объекты изображены на рис. 4.19.

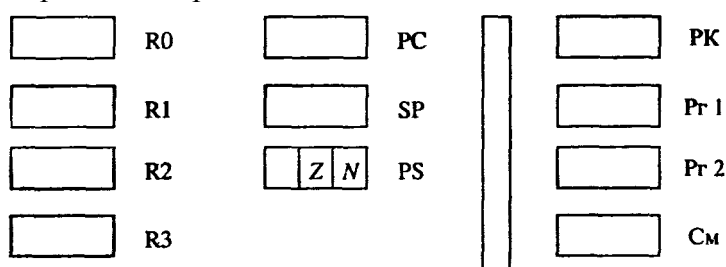


Рис. 4.19, Внутреннее устройство учебного процессора «Е97»

Кроме уже названных, в «Е97» имеется ряд *внутренних* регистров, которые процессор использует при исполнении операций. Это **регистр команд РК**, предназначенный для хранения кода исполняемой в данный момент команды; **регистры операндов Pr1 и Pr2**, куда считываются исходные данные; **сумматор См**, в котором производится требуемое в команде действие и получается результат. Мы не можем непосредственно изменять содержимое этих «служебных» регистров, но в данном учебном компьютере они доступны для наблюдения. Вертикальная заштрихованная полоса на рисунке разделяет программно-доступные (слева) и программно-недоступные (справа) регистры.

4.2. СИСТЕМА КОМАНД

Перейдем к самому важному - системе команд, которые умеет выполнять учебный процессор. Как мы уже знаем, машинная команда состоит из операционной и адресной частей: первая указывает, что надо сделать с данными, а вторая - где их взять и куда поместить результат. В этом разделе мы будем говорить, в основном, об операционной части, лишь коротко упоминая об адресной; в последней нас пока, главным образом, будет интересовать число операндов (адресов). Вопросам, связанным с подробностями адресации данных, будет посвящен следующий пункт.

Итак, рассмотрим структуру команды «Е97» (рис. 4.20). В зависимости от конкретной операции, ее формат может иметь некоторые особенности, но в наиболее полной форме он состоит из четырех частей по 4 бита каждая (см. рис. 4.20, б): **модификатор МОД**, **код операции КОП** и два **операнда ОП1 и ОП2**. Назначение КОП и операндов было описано в предыдущем параграфе. Что же касается МОД, то он указывает варианты реализации команды, например, адресовать байт или слово, по каким управляющим битам переходить и др.

Наиболее простой формат команд из всех возможных, имеют две **-нет операции** (ее код 0) и **останов** (код F). Как видно из рис. 4.20, а, в этих командах задействован только КОП. остальные 12 бит значения не имеют. Основная масса команд, коды которых заключены в интервале от 1 до В. являются двухадресными и соответствуют уже упоминавшемуся ранее рис. 4.16, б. К ним отно-

сятся:

1 - перепись,	5 - умножение,	9 - исключающее «ИЛИ»,
2 - сложение,	6 - деление,	А - ввод из порта,
3 - вычитание,	7 - логическое «И»,	В - вывод из порта.
4 - сравнение,	8 - «ИЛИ»,	

.....	КОП		

а) (КОП = 0, КОП = F)

МОД	КОП	ОП 1	ОП 2

б) (КОП = 1 - В)

МОД	КОП		ОП 2

в) (КОП = С)

.....	КОП	СМЕЩЕНИЕ

г) (КОП = D)

МОД	КОП	Д КОП	ОП 1

д) (КОП = E)

Рис. 4.20. Форматы команд учебного процессора «Е97»

Операция переписи выполняется достаточно тривиально: информация считывается из ОП1 и копируется в ОП2. Совершенно аналогично работают ввод и вывод из порта, с той лишь разницей, что в качестве одного из операндов указывается номер порта. Все остальные двухадресные команды с кодами 2-9 представляют собой определенные действия над двумя данными, выполняемые по универсальной схеме

ОП2 операция ОП 1 => ОП2

Например, по команде деления процессор извлекает ОП2, делит его на ОП1 и результат помещает вместо первоначального значения ОП2.

Некоторую особенность имеет команда сравнения. При ее исполнении производится вычитание ОП2 - ОП1, но результат никуда не записывается. («Тогда для чего же вычитать?» - спросите вы? - «Исключительно ради установки управляющих битов, которые в дальнейшем могут быть проанализированы командами условных переходов».)

Арифметические действия осуществляются над целыми числами и результаты их - целочисленные в формате «16-битные целые числа со знаком». При делении получается целая часть частного. Что же касается операций над вещественными числами, то они могут быть реализованы программным путем (соответствующие программы могут быть, например, помещены в одно из ПЗУ).

Перейдем теперь к рассмотрению команд переходов. Как мы уже знаем, они бывают *абсолютные*, когда значение адреса для перехода задается явно, и *относительные*, когда адрес следующей команды вычисляется путем значения текущего программного счетчика и указанного в теле команды смещения. В соответствии с этим в «Е97» есть два типа переходов с кодами операций С и D; их форматы представлены на рис. 4.20, в, г.

Начнем с абсолютного перехода, код которого равен С. Если процессор встретит в программе команду из двух слов

1C0D
0056

то следующей будет выполняться команда с адресом 56. Иными словами, адрес перехода берется из самой команды. В команде 1C0D старшая шестнадцатеричная цифра - модификатор, соответствующий безусловному переходу; С - код операции; 0 - неиспользуемая цифра; D - операнд, ука-

зывающий что адрес перехода хранится в следующем слове, составляющем с 1C0D единое целое. Подробности такого способа адресации - в следующем пункте.

По-другому обстоит дело с относительным переходом, код которого D. В качестве примера возьмем команду

1D06

Для определенности будем считать, что эта команда находится в памяти по адресу 42. В строгом соответствии с основным алгоритмом работы процессора, после выборки рассматриваемой команды счетчик адреса команд PC автоматически увеличивается до 44. Затем, выполняя расшифрованную команду перехода, процессор прибавит к текущему содержимому PC смещение 06 и тем самым осуществит переход на адрес $44 + 6 = 4A$. Обратите внимание, что итоговый адрес в случае относительного перехода зависит от расположения команды перехода в ОЗУ.

При отрицательном смещении возможно получение адреса меньшего, чем исходный. При обсуждении команд перехода мы незаметно включили в работу модификатор команд. Для переходов его роль проста и наглядна: МОД показывает, по какому условию осуществляется переход. Таблица всех используемых в «E97» значений модификаторов выглядит так:

- 0 - возврат из подпрограммы;
- 1 - безусловный переход;
- 2 - $N = 0 (\geq 0)$;
- 3 - $N = 1 (< 0)$;
- 4 - $Z = 0 (\neq 0)$;
- 5 - $Z = 1 (= 0)$;
- 6 - $N=1 \text{ or } Z=1 (\leq 0)$;
- '7 - $N=0 \text{ and } Z=0 (> 0)$;
- 9 - вызов подпрограммы.

Становится очевидным, что рассмотренные в обоих предыдущих примерах команды с МОД = 1 являются наиболее простым вариантом перехода - безусловным.

Для работы с условными переходами следует твердо запомнить следующее правило:

«Если анализируемое условие справедливо, т. е. состояние управляющих признаков совпадает с требуемым, то переход происходит. В противном случае никаких действий не производится, а значит переход просто игнорируется и процессор, как обычно, выбирает следующую команду».

Кстати, так обстоит дело и в языках высокого уровня.

И еще об одном виде перехода следует поговорить особо - о **переходе с возвратом** или о **переходе к подпрограмме**, как его часто называют. На практике подпрограммы (процедуры) и функции играют очень важную роль. Подпрограммы полезны, когда *в разных местах* программы требуется выполнить *одни и те же действия*. В этом случае имеет смысл оформить повторяющиеся действия в виде подпрограммы, а затем просто вызывать ее в нужных местах. В каком-то смысле это похоже на публикацию текстов песен, когда припев пишется один раз, а в дальнейшем просто ставится ссылка на него в виде слова «ПРИПЕВ».

Наиболее важное отличие перехода к подпрограмме от безусловного перехода состоит в том, что требуется иметь возможность вернуться из подпрограммы в то же самое место, откуда она была вызвана. Применительно к процессору возможность возврата означает запоминание где-нибудь значения программного счетчика PC; для возврата достаточно будет просто восстановить в PC сохраненное значение.

Для обеспечения возможности вложения друг в друга подпрограмм необходимо уметь сохранять не одно значение PC, а несколько. Для реализации такого механизма памяти используется стек, который идеально подходит для любой вложенности конструкций и при этом требует наличия всего одного выделенного регистра-указателя стека SP.

Рассмотрим конкретный пример. Пусть в некотором месте программы находится команда из двух слов

9C0D

0030

Исполнять эту команду процессор будет так. Прежде всего, он уменьшит SP на 2 и запомнит по полученному адресу текущее содержимое PC (вспомните, что счетчик к этому времени уже

будет показывать на следующую команду). Затем последует переход по адресу 30, считанному ранее из второго слова команды. Таким способом мы попадем в подпрограмму, надежно спрятав в стеке адрес основной программы, куда нужно вернуться. Обсудим теперь, как произойдет возврат.

В конце любой подпрограммы должна стоять команда 0C00 или 0D00. Встретив ее, процессор извлечет из стека занесенное туда ранее значение и поместит его в РС. При этом он увеличит SP на два, освободив ненужную более ячейку памяти в стеке. Таким образом, прерванное на время выполнение основной программы продолжится с нужного места.

Примечание. Команды с кодами 0C и 0D тождественны и их младшие 8 бит не используются.

Для того, чтобы лучше разобраться в описанном механизме, попробуйте самостоятельно проанализировать ситуацию, когда одна подпрограмма вызывает другую.

Займемся последней группой команд процессора, для которой КОП = E. Прежде всего, почему такому КОП соответствует группа, а не одна команда? Все дело в том, что эти команды одноадресные, и освобождающиеся от одного из операндов 4 бита можно использовать для задания номера операции в этой группе. Назовем полученные биты дополнением к КОП - ДКОП. В итоге получим формат, приведенный на рис.4.20, д). Из рисунка видно, что код операций для всех одноадресных операций состоит из двух шестнадцатеричных цифр, причем первая из них всегда E. Познакомимся с этими командами более подробно.

Команда с кодом E1 выполняет над единственным операндом ОП1 логическую операцию НЕ, т.е. заменяет нулевые биты единицами и наоборот (инверсия).

Команды с кодами E2 - E9 обеспечивают работу со стеком. Так, при коде операции E2 ОП1 заносится в стек, а при E3 - считывается оттуда. Например, вот как можно поменять местами содержимое регистров R1 и R2 с использованием стека:

0000	0E21	записать R1 в стек
0002	0E22	записать R2 в стек
0004	0E31	считать значение из стека в R1
0006	0E32	считать значение из стека в R2
0008	0F00	останов

Команды E4 и E5 позволяют изменять значение SP на величину ОП1, что часто бывает полезно при работе со стеком, например, при освобождении в нем сразу нескольких «этажей». По кодам E6 и E7 можно задать новое значение SP и прочитать его текущее значение в ОП1. Наконец, наиболее экзотические из этой группы команды E8 и E9 сохраняют в стеке и восстанавливают для последующего анализа регистр состояния процессора PS. Эти команды замечательны тем, что обрабатывают вполне определенный операнд, поэтому содержимое ОП1 в команде значения не имеет; *договоримся заполнять его нулем.*

Осталось рассмотреть последнюю группу команд - сдвиги. Их коды EA - EC. Все они осуществляют сдвиг кода в ОП1 на один разряд влево или вправо в зависимости от значения ДКОП. Полезно помнить, что сдвиг влево эквивалентен умножению, а вправо - делению на два.

Команда EC, называемая арифметическим сдвигом, отличается от обычного сдвига EB тем, что старший знаковый разряд при арифметическом сдвиге сохраняет свое значение, например:

Исходное значение ОП1: 1111 0000 1111 0000

Результат команды EB: 0111 1000 0111 1000

Результат команды EC: 1111 1000 0111 1000.

Арифметический сдвиг бывает полезен для деления отрицательных чисел, так как в этом случае автоматически сохраняется признак знака минус - единица в старшем разряде.

В данной модели при сдвиге, приводящем в выходу старшей или младшей цифры за пределы разрядной сетки, соответствующая информация теряется. В реальных процессорах для ее сохранения существует специальный управляющий бит, называемый «битом переноса».

4.3. АДРЕСАЦИЯ ДАННЫХ

Теперь, когда мы знаем практически все об операционной части команды, можно заняться адресной. Посмотрим, какими способами могут представляться операнды ОП1 и ОП2.

Начнем с того, что под кодирование каждого операнда всегда отводится четыре двоичных разряда. Из них старшие два будут всегда задавать тип адресации данных, а младшие - его конкретизировать. В большинстве случаев два младших бита будут просто представлять собой номер регистра, с участием которого осуществляется адресация.

Старшая «половинка» модификатора операнда, соответствующая типу адресации, может содержать четыре неповторяющиеся двоичные комбинации:

00 - *регистровый* метод адресации: операнд является содержимым указанного регистра;

01 - метод *косвенной* адресации: операндом является содержимое ячейки ОЗУ, адрес которой задан в указанном регистре;

10 - резерв; в будущей версии здесь будет реализован индексный метод адресации;

11 - адресация *по счетчику адреса команд* РС: операнд извлекается с использованием информации, входящей в команду (более подробные объяснения приведены ниже).

Рассмотрим перечисленные способы адресации подробнее.

Наиболее простым является регистровый метод, который мы фактически уже использовали в примерах предыдущего раздела. Учитывая, что этому методу соответствуют нулевые значения старших битов, полный код операнда совпадает с номером регистра: двоичная комбинация 0000 соответствует R0, 0001 - R1 и т.д.

В качестве данных для операции используется информация, содержащаяся в указанном регистре. Например, если R1 = 3, а R2 = 5, то в результате выполнения команды

0212:R2+R1=>R2

получится R2 = 8.

При косвенной адресации код операнда выглядит несколько сложнее: косвенное обращение по регистру R0 имеет вид 0100 (т.е. 4), по R1 - 0101 (5) и т.д. Содержимое указанного регистра при этом служит не операндом, а его адресом в ОЗУ. Рассмотрим команду

0263: R3 + (R2) => R3

где скобки у R2 символизируют косвенную адресацию. Пусть содержимое R2 в данный момент равно 30, а R3 = 6. Примем также, что в ячейке памяти с адресом 30 хранится число 10. Тогда процессор «Е97», выполняя команду, к имеющемуся в R3 значению 6 прибавит число из ячейки 30, на которую указывает R2, и результат операции - 16 - занесет в R3.

И, наконец, рассмотрим способы адресации по программному счетчику РС. Поскольку в этом случае регистр, по которому производится адресация, уже однозначно определен, освобождаются два младших бита операнда, которые можно использовать для других целей. В связи с этим удастся получить четыре различных способа адресации по счетчику:

11 00 - резерв;

11 01 - операндом служит константа, входящая в команду;

11 10 - операнд извлекается из ячейки ОЗУ, адрес которой входит в команду;

1111-резерв.

Итак, в «Е97» существует два метода адресации по РС, соответствующих шестнадцатеричным кодам **D** и **E** в качестве операнда. Изучим их на примерах. Команда, состоящая из двух слов

02D1

0020

выполняет операцию $R1 + 20 \Rightarrow R1$ следующим образом: к текущему значению R1 прибавляется извлеченное из команды число 20 и результат помещается в R1. Если первоначальное значение R1 было, скажем, F0, то в результате операции в R1 запишется 110.

Рассмотрим еще одну команду с адресацией по РС:

021E

0020

выполняющую операцию $(20) + R1 \Rightarrow (20)$ так: к числу, хранящемуся в ячейке памяти 20, прибавляется значение R1. В случае, когда R1 = F0, а (20) = 40, ответ будет: $(20) = 130$.

На этом рассказ о методах адресации можно было бы и закончить, если бы не наличие в командах модификатора, хранящегося в первой цифре шестнадцатеричного представления команды. Его значение также может существенно влиять на извлечение данных и запись результата.

Во всех приведенных в данном пункте примерах МОД = 0 и, следовательно, не оказывает влияния на расшифровку и выполнение операции. Если же он имеет ненулевое значение, необхо-

димо дополнительно учитывать и этот фактор.

Модификатор состоит из четырех битов, причем два старших отвечают за «переключением бант/слово», а два младших - за особый способ представления данных, называемый «**короткой константой**». Многие команды оперируют с небольшими константами, значения которых не превышает по модулю 15. Такие константы вполне «уместятся» в четырех отведенных под ОП1 битах. Для реализации такой возможности в «Е97» используются два младших бита модификатора - 0-й и 1-й. Признаком, по которому процессор распознает «короткую константу», служит наличие единицы в первом разряде модификатора. В этом случае содержимое ОП1 рассматривается как двоичное число, а 0-й бит модификатора служит его знаком, например:

$$25A3:R3*10=>R3$$

или

$$3653:R3/(-5)=>R3$$

Из приведенных примеров видно, что операции с «короткой константой» занимают всего одно слово. Данный способ оказывается, таким образом, экономичнее описанного чуть раньше метода адресации по РС (вспомните, что там константа хранится во втором слове команды).

Наличие операций с «короткой константой» позволяет упростить систему команд процессора. Например, в программах часто требуются команды, увеличивающие или уменьшающие данные на единицу. Любой процессор имеет для этой важной ситуации специализированные команды, обозначаемые чаще всего INC и DEC. В «Е97» эти команды получаются как частный случай операций с «короткой константой».

Во всех предыдущих рассуждениях мы молчаливо полагали, что процессор работает только с 16-разрядными словами. Но это не всегда так - иногда требуется выполнить те или иные действия над байтами, например, при обработке текста. Старшие биты модификатора как раз и предусмотрены для оказания режима байт/слово: 3-й бит соответствует ОП1, а 2-й - ОП2. Если соответствующий бит модификатора установлен в единицу, то процессор оперирует с байтом, иначе - со словом. Как всегда, наличие двух битов порождает четыре варианта:

- 00 ОП1-слово, ОП2-слово;
- 01 ОП1-слово, ОП2-байт;
- 10 ОП1 - байт, ОП2 - слово;
- 11 ОП1-байт, ОП2-байт.

Наиболее простыми и очевидными являются комбинации слово - слово и байт -байт, когда оба операнда и результат имеют одинаковый тип. Смешанные комбинации слово - байт и байт - слово более экзотичны и нуждаются в особом рассмотрении. Не вдаваясь в тонкости, можно считать, что в двухадресных командах модификатор всегда равен либо **0 для операции со словами**, либо **С для байтов**.

4.4. РАБОТА С ВНЕШНИМИ УСТРОЙСТВАМИ

Как уже говорилось ранее, модель «Е97» отражает наиболее простые методы обмена с внешними устройствами из тех, что свойственны современным компьютерам.

Как и в реальных микропроцессорах, «общение» с периферийными устройствами осуществляется через порты ввода-вывода. Для учебной ЭВМ принята наиболее простая модель, когда каждому из имеющихся внешних устройств соответствует два порта - **порт данных** и **порт состояния**. Можно считать, что каждый порт представляет собой 16-разрядный регистр, находящийся в контроллере. Через порт данных происходит обмен информацией. Порты состояния позволяют управлять процессом обмена данными, так как предоставляют процессору сведения о текущем состоянии внешнего устройства. Реальная синхронизация ввода-вывода является достаточно сложной проблемой и существенно зависит от типа периферийного устройства. Поэтому в «Е97» регистр состояния устроен максимально просто: в нем имеется единственный используемый бит - **бит готовности**. В обсуждаемом учебном компьютере бит готовности помещается в

седьмом разряде регистра состояния.

Все порты являются 16-разрядными, но реальная информация всегда располагается в их младшем байте. Содержимое битов с 8-го по 15-й в выходные порты формально заносится, но никакого влияния на устройство вывода не оказывает.

Каждому порту соответствует свой номер (адрес). В «Е97» из шестнадцати возможных используются следующие порты:

- 0 - порт состояния клавиатуры (только чтение);
- 1 - порт данных клавиатуры (только чтение);
- 2 - порт состояния дисплея (только чтение);
- 3 - порт данных дисплея (только запись).

Стандартный алгоритм обмена с использованием портов ввода-вывода выглядит следующим образом. Считывается порт состояния и проверяется значение *знакового бита* его *младшего* бита. Эта операция повторяется до тех пор, пока бит готовности не будет установлен внешним устройством в единицу, что является сигналом процессору к началу обмена. Только после него процессор может записать информацию в порт данных, если речь идет об устройстве вывода, или считать их из порта, если это порт ввода.

Вот как может выглядеть правильная программа вывода одиночного символа из R0 на дисплей (именно так она реализована в ПЗУ «Е97»):

```
0A21 порт2=>R1
E401 сравнить с 0 R1b
2DFA если ≥ 0, PC - 6 (к повторению проверки)
0B03 R0 => порт 3
```

Первые три команды считывают и проверяют бит готовности в порту 2, а последняя команда обеспечивает собственно вывод требуемого символа на дисплей.

Обратите внимание на команду с «короткой константой» 0, которая **обязательно** должна проверять знаковый разряд *младшего байта*, а не всего слова в целом.

Мы подробно рассмотрели программирование процедуры вывода символа; ввод осуществляется совершенно аналогично.

4.5. ПРИМЕРЫ ПРОГРАММ

Пример 1. Вычислить полную поверхность параллелепипеда со сторонами A, B и C. Считать, что исходные значения находятся в ячейках ОЗУ. Результат также поместить в ячейку памяти.

Решение. Полная поверхность параллелепипеда вычисляется по формуле

$$S=2*(A*B+A*C+B*C)$$

Для упрощения программы выражение удобно представить в виде $S = 2 * [A * (B + C) + B * C]$

Программа 134

0000	01E0	(22) => R0	B => R0
0002	0022		
0004	0101	R0 => R1	B => R1
0006	02E0	R0 + (24) => R0	B + C
0008	0024		
000A	05E0	R0 * (20) => R0	A * (B + C)
000C	0020		
000E	05E1	R1 * (24) => R1	B * C
0010	0024		

0012	0210	$R0 + R1 \Rightarrow R0$	$A * (B + C) + B * C$
0014	0200	$R0 + R0 \Rightarrow R0$	$2 * [A * (B + C) + B * C]$
0016	010E	$R0 \Rightarrow (26)$	результат $\Rightarrow S$
0018	0026		
001A	0F00	останов	
.....			
0020	0002	A	
0022	0003	B	
0024	0004	C	
0026	0034	S	

Примечание. Не забывайте, что ответ 34 в памяти ЭВМ представлен в шестнадцатеричной системе. В десятичной, как и положено, получится 52.

Пример 2. Организовать ввод латинских букв таким образом: программа принимает латинскую букву и обрабатывает ее так, чтобы она всегда была заглавной.

Решение. Главная «хитрость» решения состоит в том, чтобы понять, чем отличаются заглавные буквы от строчных. Для этого из таблицы ASCII, приведенной в главе 1, выберем наугад одну из букв и выпишем в двоичном виде коды заглавного и строчного символов. Например, для буквы R получим

R 0101 0010
r 0111 0010

Повторив аналогичные действия еще для нескольких букв, сделаем вывод, что **коды заглавных и строчных букв отличаются единственным битом - пятым**, если считать младший бит за нулевой. Запомним этот факт, поскольку он справедлив для любого современного компьютера.

Итак, для нашей задачи требуется сохранить все биты, кроме пятого, значит потребуется выполнить логическую операцию **И** с константой

1101 1111 =DF,

после чего любой код «потеряет» ненужный нам бит, сохраняя все остальные.

Еще одной особенностью решения является активное использование подпрограмм из ПЗУ. Поскольку при вызове подпрограмм обязательно используется стековая память, то должен быть корректно определен указатель стека SP; программа начинает работу с установки указателя стека.

Программа 135

0000	0E6D	26 $\Rightarrow$ SP	установка указателя стека
0002	0026		
0004	9C0D	вызов п/п 40FE	ввод символа (без эхо-печати!)
0006	40FE		
0008	0101	$R0 \Rightarrow R1$	сохранить введенный символ
000A	07D0	DF and $R0 \Rightarrow R0$	сделать букву заглавной
000C	00DF		
000E	9C0D	вызов п/п 4088	вывести результат
0010	4088		
0012	0F00	останов	

Пример 3. Найти максимум из трех чисел, находящихся в регистрах R1, R2 и R3. Переписать наибольшее из них в R0.

Решение. Сначала максимальное из чисел в R1 и R2 занесем в R0. После этого, если R3 окажется больше R0, «исправим положение», переписав в качестве ответа R3.

Программа 136

0000	0412	сравнить R2 с R1	
0002	3D04	если < 0 , то $PC=PC+4$	к записи R1 (к 0008)
0004	0120	$R2 \Rightarrow R0$	запомнить R2

0006	1D02	PC=PC+2	обход второй ветви (к 000A)
0008	0110	R1 => R0	запомнить R1
000A	0403	сравнить R3 с R0	
000C	3D02	если < 0, то PC=PC+2	к выходу (к 0010)
000E	0130	R3 => R0	запомнить R3
0010	0F00	останов	

Пример 4. Найти сумму первых 100 натуральных чисел.

Говорят, такую задачу некогда решил в уме юный Гаусс, когда учился в школе. Мы будем решать задачу «в лоб», т.е. честно суммировать с помощью компьютера. *Решение.* Поместим в R3 обрабатываемое в данный момент число N (меняется от 1 до 100), а в R0 - результирующую сумму S. Зададим им начальные значения и будем циклически добавлять к S текущее значение N. Признаком окончания цикла будет ситуация, когда $N > 100$.

Программа 137

0000	2113	1 => R3	1 => N
0002	2100	0 => R0	0 => S
0C04	0230	R0 + R3 => R0	S = S + N
0006	2213	R3 + 1 => R3	N=N+1
0008	04D3	сравнить R3 с 100	сравнить N и 100
000A	0064		
000C	6DF6	если ≤ 0 , то PC=PC+F6	если $N \leq 100$,
		к повторению (000E + FFF6 = 0004)	
000E	0F00	останов	

Пример 5. Вывести на экран весь латинский алфавит от A до Z. *Решение.* Разместим в R0 выводимый латинский символ, первоначальное значение которого будет «A» (код 65 = 41h). Вывод будем осуществлять обращением к соответствующей подпрограмме ПЗУ. Для перехода к следующему символу алфавита достаточно прибавить 1 к коду текущего символа (очень похоже на переход к следующему числу в предыдущем примере). Остается только проверить, не выходит ли вновь полученный символ за латинский алфавит, т.е. не превышает ли его код 5Ah («Z»), и, если ответ будет «да» (превышает), то закончить процедуру.

Программа 138

0000	0E6D	26 => SP	установка указателя стека
0002	0026		
0004	01D0	41 => R0	код первого символа
0006	0041	(«A»)	
0008	9C0D	вызов п/п 4088	вывод символа
000A	4088		
000C	2210	R0 + 1 => R0	следующий символ
000E	04D0	сравнить R0 с 5A его код $\leq$ «Z»?	
0010	005A	(«Z»)	
0012	6DF4	если ≤ 0 , то PC=PC+F4	к повторению (0008)
0014	0F00	останов	

Пример 6. В памяти, начиная с адреса 001A, хранится некоторый текст, длина которого равна 15 (Fh) байтам. Определить номер первого, совпадающего с образцом, символа в тексте. При отсутствии требуемого символа результат равен 0 (это практически полный аналог функции POS в Паскале).

Решение. Поместим в R1 счетчик символов, в R2 - адрес текущего символа. Затем будем сравнивать каждый символ текста с образцом в R0 и в случае совпадения прерываем выполнение цикла. При несовпадении будем продолжать цикл до тех пор, пока счетчик не превысит Fh, т.е. не

станет равным 10h Если цикл завершится по выполнении этого условия, то символ-образец найти не удалось и в качестве ответа в R1 следует занести 0.

Программа 139

0000	2111	1 => R1	номер символа
0002	01D2	1A => R2	адрес начала текста
0004	001A		
0006	C460	сравнить R0b с (R2)b	сравнить символ с образцом
0008	5D0C	если = 0, то PC=PC+2	выход при совпадении (к 0016)
000A	2211	R1 + 1 => R1	увеличить номер символа
000C	2212	R2 + 1 => R2	вычислить следующий адрес
000E	04D1	сравнить R1 с 10	текст не закончился?
0010	0010		
0012	4DF2	если ≠ 0, то PC=PC+F4	нет - к повторению (к 0006)
0014	2101	0 => R1	при отсутствии символа - 0
0016	0F00	останов	
0018	0000		
001A	4854	«ТН»	текст
001C	5349	«IS»	«THIS IS MY TEXT»
001E	4920	«I»	
0020	2053	«S »	
0022	594D	«MY»	
0024	5420	«T»	
0026	5845	«EX»	
0028	0054	«T»	

4.6. НЕКОТОРЫЕ СПРАВОЧНЫЕ ДАННЫЕ ПО E-97

А. Система команд процессора

МОД	КОП	ОП1	ОП 2	Пояснения
X	0	X	X	Нет операции
X	1	X	X	оп 1 => оп 2
X	2	X	X	оп 2 + оп 1 => оп 2
X	3	X	X	оп 2 – оп 1 => оп 2
X	4	X	X	оп 2 + оп 1 (сравнить)
X	5	X	X	оп 2 * оп 1 => оп 2
X	6	X	X	оп 2/оп 1 => оп 2
X	7	X	X	оп 2 AND оп 1 => оп 2
X	8	X	X	оп 2 OR оп 1 => оп 2
X	9	X	X	оп 2 XOR оп 1 => оп 2
X	A	X	X	порт! => оп 2
X	B	X	X	оп 1 => порт 2
X	C	X	X	Переход по адресу
X	D	X	X	Переход по смещению
X	E	*	X	(Одноадресные операции)
X	F	X	X	Останов

МОД	КОП	ОП1	Пояснения
X	E1	X	NOT оп 1

X	E2	X	оп 1 => стек
X	E3	X	стек => оп 1
X	E4	X	SP + оп 1 => SP
X	E5	X	SP – оп 1 => SP
X	E6	X	Оп 1 => SP
X	E7	X	SP => оп 1
X	E8	0	PS => стек
X	E9	0	стек => PS
X	EA	X	Сдвиг влево оп 1
X	EB	X	Сдвиг вправо оп 1
X	EC	X	Арифметический сдвиг вправо оп 1

Б. Двоичные коды модификатора МОД для переходов

0000	-	возврат из подпрограммы
0001	-	безусловный переход
0010	-	N=0 (>=0)
0011	-	N=1 (<0)
0100	-	Z=0 (<>0)
0101	-	Z=1 (=0)
0110	-	N=1 or Z=1 (<=0)
0111	-	N=0 and Z=0 (>0)
1001	-	вызов подпрограммы.

В. Важные кочанды с «короткой константой»

XX 10 0001 0000 XXXX	-	очистить оп 1
XX 10 0010 0001 XXXX	-	+1 в оп 1
XX 11 0010 0001 XXXX	-	*****
или		
XX 10 0011 0001 XXXX	-	-1 из оп 1
XX 10 0100 0000 XXXX	-	сравнить 0 с оп 1
XX 11 0101 0001 XXXX	-	*(-1) оп1
XX 10 0111 0001 XXXX	-	оп 1 MOD 2 => оп 1

Г. Кодирование операндов

0	0000	R0		1000	резерв
1	01	R1		01	резерв
2	10	R2		10	резерв
3	11	R3		11	резерв
4	0100	(R0)		1100	резерв
5	01	(R1)		01	константа
6	10	(R2)		10	адрес ОЗУ
7	11	(R3)		11	резерв

Контрольные вопросы и задания

1. Назовите основные функциональные устройства учебного компьютера «Е97» и охарактеризуйте их назначение.
2. Какие виды памяти существуют в «Е97» и как они адресуются?

3. Сопоставьте адресацию к байтам и к словам. Какие вам удалось заметить различия?
4. В ОЗУ, начиная с адреса 300, хранятся следующие байты: 00, 01, 02, 03. Какое слово будет прочитано по адресу 302?
5. В ОЗУ начиная с адреса 300 хранятся слова 0102 и 5301. Совпадают ли находящиеся в адресах 301 и 302 байты?
6. Что такое BIOS и какова его роль в современном компьютере? Знаете ли вы название фирмы, написавшей BIOS для вашего компьютера?
7. Какие значения получают управляющие признаки, если результат операции получится равным 1?0?
8. Какие внутренние регистры содержит учебный процессор «Е97» и каково их назначение?
9. Из каких частей состоит полная команда «Е97»?
10. Какие форматы может иметь команда «Е97»? Приведите примеры команд на каждый из форматов.
11. Напишите программы вычисления нескольких арифметических выражений.
12. Чем различаются переходы с кодами С и D и как они работают?
13. Если рассмотренную в одном из примеров команду 1D06 поместить в ячейку с адресом не 42, а 4E, то какая команда будет выполняться следующей?
14. Каково должно быть смещение в команде перехода к адресу 56, если сама команда имеет адрес 4E? 5E?
15. Какие значения может принимать модификатор в командах перехода и что они обозначают?
16. Напишите программу, которая переписывает в R3 наименьшее из чисел, хранящихся в регистрах R1 и R2.
17. Придумайте задачи, при программировании которых может быть полезно использование процедур или функций.
18. Как процессор обращается к подпрограмме и выходит из нее обратно?
19. Какие операции можно делать с указателем стека?
20. Чем отличаются сдвиги с кодами операций EB и EC? Приведите пример, когда эти различия незаметны.
21. Какие методы адресации реализованы в «Е97»? Приведите по два-три примера на каждый из них.
22. Как кодируются следующие операнды: R2? (R3)? константа 345? ячейка 66?
23. Объясните, как будет выполняться команда 0572 и какие значения требуется знать, чтобы предсказать ее результат.
24. Чем различается выполнение команд 0301 и 0345?
25. Изменится ли значение регистра R0 при выполнении команды 0E14?
26. Сколько слов занимает команда (40) - (50) Ю (40) и как она кодируется?
27. Как записывается команда, умножающая на 3 содержимое ячейки 50?
28. Какой командой можно присвоить регистру R2 некоторое значение?-
29. Что такое «короткая константа» и когда она бывает полезна?
30. Для команд 25A3 и 3653 напишите эквивалентные, используя метод адресации констант по счетчику команд.
31. Почему среди списка полезных команд с «короткой константой» нет умножения на 2?
32. Подробно объясните с точностью до бита кодировку команды C470. ,,
33. Что такое порт ввода-вывода?
34. Как называются порты, имеющиеся у каждого из внешних устройств «Е97»?
35. Перечислите все известные вам порты ввода-вывода, реализованные в учебном компьютере «Е97». и их номера.
36. Каков стандартный алгоритм обмена процессора с внешним устройством? Какую роль играет в нем бит готовности?
37. По аналогии с приведенной в тексте программой вывода символа на дисплей, напишите самостоятельно программу ввода символа с клавиатуры. Не забудьте организовать при этом «эхо-печать», т.е. вывести принятый с клавиатуры символ.

§ 5. ВНЕШНИЕ УСТРОЙСТВА ЭВМ: ФИЗИЧЕСКИЕ ПРИНЦИПЫ И ХАРАКТЕРИСТИКИ

Внешние (или, по другому, периферийные) устройства ЭВМ прошли огромный путь в своем развитии. Существуя машина времени, инженеры-конструкторы и пользователи ЭВМ 50-х годов, увидев принтер, вряд ли поверили бы, что этот миниатюрный настольный аппарат может делать гораздо больше, чем его предок ростом с человека - АЦПУ (автоматическое цифровое печатающее устройство), и вообще не поняли бы, чем занимается сканер или «мышь». Возможности компьютера в значительной степени определяются номенклатурой и производительностью внешних устройств.

5.1. ВНЕШНИЕ ЗАПОМИНАЮЩИЕ УСТРОЙСТВА

Внешние запоминающие устройства (ВЗУ) обеспечивают долговременное хранение программ и данных. Наиболее распространены следующие типы ВЗУ: накопители на магнитных дисках (НМД); их разновидности - накопители на гибких магнитных дисках (НГМД) и накопители на жестких магнитных дисках (НЖМД); накопители на магнитных лентах (НМЛ); накопители на оптических дисках (НОД).

Соответственно, физическими носителями информации, с которыми работают эти устройства, являются магнитные диски (МД), магнитные ленты (МЛ) и оптические диски (ОД).

Принцип записи информации на магнитных носителях основан на изменении намагниченности отдельных участков магнитного слоя носителя (диска, ленты). Запись осуществляется с помощью магнитной головки: электрические сигналы, возникающие под управлением электронного блока, возбуждают в ней магнитное поле, воздействующее на носитель и оставляющее намагниченные участки на заранее размеченных дорожках. При считывании информации эти намагниченные участки индуцируют в магнитной головке слабые токи, которые превращаются в двоичный код, соответствующий ранее записанному.

Накопители на магнитных дисках включают в себя ряд систем:

- электромеханический привод, обеспечивающий вращение диска;
- блок магнитных головок для чтения-записи;
- системы установки (позиционирования) магнитных головок в нужное для записи или чтения положение;
- электронный блок управления и кодирования сигналов.

НГМД - устройство со сменными дисками (их часто называют «дискетами»). Несмотря на относительно невысокую информационную емкость дискеты, НГМД продолжают играть важную роль в качестве ВЗУ, поскольку поддерживают ряд функций, которые не обеспечивают другие накопители. Среди них отметим

- возможность транспортировки информации на любые расстояния;
- обеспечение конфиденциальности информации (дискету можно положить в карман сразу после окончания сеанса работы).

Дискета - гибкий тонкий пластиковый диск с нанесенным (чаще всего на обе стороны) магнитным покрытием, заключенный в достаточно/твердый и - картонный или пластиковый - конверт для предохранения от механических повреждений. Информация на диск наносится вдоль концентрических окружностей - дорожек. Каждая дорожка разбита на несколько секторов (обычно 9 или 18) - минимально возможных адресуемых участков. Стандартная емкость сектора - 512 байт. На двухсторонней дискете две одинаковые дорожки по обе стороны диска образуют цилиндр. Процедура разметки нового диска - нанесение секторов и дорожек - называется форматированием. Иногда приходится прибегать к переформатированию диска, на котором уже есть информация; последняя в таком случае практически обречена на уничтожение.

Тип дискеты обычно указывается на ее конверте:

- DS (double side) - двухсторонняя;
- DD (double density) - двойной плотности;
- HD (high density) - высокой плотности.

Возможны сочетания типа DS/DD, DS/HD и др.

Стандартные размеры (диаметры) дискет 133 мм (5,25 дюйма; постепенно выходят из экс-

плутации) и 89 мм (3,5 дюйма). Появились, но пока не получили широкого распространения, дискеты диаметром 51 мм.

Важнейшая, с точки зрения пользователя, характеристика дискеты - информационная емкость. Чаще всего она находится в диапазоне от одного до полутора мегабайт, хотя созданы дискеты с емкостью до 10 Мбайт. Специальные дискеты для резервного копирования (так называемые Zip-дискеты, для работы с которыми нужны особые дисководы) имеют емкость 100 Мбайт и более. Другие важнейшие характеристики - скорость доступа к определенному участку информации и скорость записи или считывания информации - определяются не столько самой дискетой, сколько возможностями НГМД. Доступ к информации осуществляется за время в диапазоне от 0,1 с до 1 с (что очень велико по сравнению с другими типами дисководов), скорость чтения/записи порядка 50 кбайт/с, что по современным представлениям весьма немного.

Жесткий диск сделан из сплава на основе алюминия и также покрыт магнитным

слоем. Он помещен в неразборный корпус, встроенный в системный блок компьютера. По всем профессиональным характеристикам жесткие диски (и соответствующие накопители) значительно превосходят гибкие: емкость от 20 Мбайт до 10 Гбайт (реально диски с емкостью меньшей, чем 1 Гбайт, давно не выпускаются), время доступа к конкретной записи в диапазоне от 1 до 100 миллисекунд (мс), скорость чтения/записи порядка 1 Мбайта/с. Скорость вращения дисков велика, обычно 3600 об/мин, что и обеспечивает относительно короткое время доступа. Однако, жесткий диск не предназначен для транспортировки информации, и это не позволило накопителям на жестких дисках вытеснить НГМД.

Первые **накопители на оптических дисках** появились в начале 70-х годов, но широкое распространение получили значительно позже. Существует несколько разновидностей оптических дисков, предназначенных для устройств, допускающих только чтение (CD-ROM, т.е. Compact Disk Reed Only Memory - компакт-диск только для чтения), для устройств, допускающих хотя бы однократную запись информации на рабочем месте пользователя и для устройств, позволяющих, подобно накопителям на магнитных дисках, многократную перезапись информации. CD-ROM диск, запись на который производится один раз при его создании и не может быть изменена, представляет собой прозрачную поликарбонатную (вид стекла) пластинку, одна сторона которой покрыта тончайшей алюминиевой пленкой, играющей роль зеркального отражателя, поверх которой нанесен защитный слой лака. Информация на ней представляется подобно тому, как на старых грампластинках - чередованием углублений и пиков, однако не в аналоговом, а в цифровом (двоичном) коде. Этот рельеф создается при производстве механическим путем (контактом с твердой пластинкой - матрицей). Информация наносится вдоль тончайших дорожек (радиальная плотность записи более 6000 дорожек/см, что в несколько десятков раз больше, чем для гибкого диска). Считывание информации осуществляется путем сканирования дорожек лазерным лучом, который по-разному отражается от углублений и пиков (по этому отражению восстанавливается записанный двоичный код). Вдоль дорожек оптического диска со скоростью 200 - 500 раз в минуту пробегает лазерный луч. При создании дисков, позволяющих вести многократную перезапись, доминирует магнито-оптический принцип (CD-MO диски). В основу положен следующий физический принцип: коэффициент отражения лазерного луча от по-разному намагниченных участков диска с особым образом нанесенным магнитным покрытием различен. Таким образом запись на MO-диски магнитная, а считывание - оптическое (лазерным лучом).

Профессиональные характеристики оптических дисков, в общих чертах таковы: емкость записи и скорость доступа к информации того же порядка, что у жестких дисков. По надежности хранения информации оптические диски в настоящее время не имеют себе равных.

Все вместе взятое и определяет место НОД в современном мире информационных технологий: от очень важных, но все-таки факультативных, устройств они становятся обязательной принадлежностью компьютеров. По мере снижения стоимости оборудования CD-MO диски могут вытеснить гибкие магнитные диски, так как, обладая значительно превосходящими профессиональными характеристиками, обеспечивают все функции ГМД. Заметим, что ситуация в этой области меняется чрезвычайно быстро.

Накопители на магнитных лентах имели огромное значение для ЭВМ первых поколений. Собственно, поначалу кроме них надежных накопителей информации большой емкости вообще не было. По мере развития ЭВМ НМЛ оттеснялись на периферию в списке ВЗУ, но свое устойчивое место занимают по сей день (хотя пользователям персональных компьютеров это не

очень заметно). Ясно, что по скорости доступа к информации НМЛ всегда будут многократно проигрывать дисковым накопителям - ведь для того, чтобы считать информацию на некотором месте ленты, необходимо отмотать предшествующий ее кусок с начала. Однако по-прежнему на лентах хранят большие объемы информации, которая не является оперативной, но требует очень надежного хранения, а также конфиденциальности.

На персональных компьютерах иногда используют специальный кассетный накопитель, размеры которого совпадают с размерами НГМД и который можно вставить на место последнего - так называемый *стриммер*. Он удобен, например, для переноса информации с жесткого диска одного компьютера на другой, долговременного хранения особо ценных системных и личных программ и данных.

5.2. УСТРОЙСТВА ВВОДА ИНФОРМАЦИИ

Разумеется, для ввода (и вывода) информации используются все виды ВЗУ. Заметим, что информация в ВЗУ хранится в виде, недоступном для непосредственного восприятия человеком, ибо ВЗУ предназначены для промежуточного хранения данных.

Многие другие устройства ввода/вывода, напротив, предназначены для обмена информацией с человеком.

Важнейшим из устройств ввода, несомненно, является **клавиатура**. Подавляющее большинство современных клавиатур являются полноходовыми контактными, т.е. клавиша утапливается при нажатии и замыкает контакт между двумя металлическими пластинками, покрытыми, во избежание окисления, пленкой благородного металла. Хорошая клавиатура рассчитана на несколько десятков миллионов нажатий каждой клавиши. При нажатии клавиши генерируется связанный с ней код, заносимый в соответствующий буфер памяти, а при ее отпускании - другой код, что позволяет перепрограммировать назначение клавиш, вводя новую таблицу соответствия этих кодов.

Ряд клавиш при совместном нажатии пары из них генерируют специальный код, отличный от того, который генерируется при нажатии каждой клавиши в отдельности. Это позволяет значительно увеличить возможности клавиатуры. Вспомним, что для передачи всех возможностей при байтовой системе кодирования могло бы понадобиться 256 клавиш, чего нет ни на одной клавиатуре благодаря указанным совмещениям.

Большинство клавиатур имеют стандартные группы клавиш:

- клавиши пишущей машинки - для ввода букв, цифр и других знаков;
- служебные клавиши, перенацеливающие действия остальных (переключатели регистров, переходы с латинского шрифта на русский и другие);
- функциональные клавиши F1 - F12 (иногда их меньше), назначение которых задает разработчик прикладной программы;
- дополнительные цифровые клавиши для большего удобства в работе.

Важным свойством клавиатуры, благодаря которому пользователь может работать не один час подряд, является эргономичность. Этим термином задается совокупность характеристик, определяющих удобство (в широком смысле слова) устройства. По отношению к клавиатуре это дизайн, отсутствие бликов, удобное взаиморасположение и размеры клавиш и многое другое.

В состав стандартного оснащения современного персонального компьютера входит **мышь** - устройство ввода и управления. Конструктивно это коробочка с выступающим внизу шариком, который, поворачиваясь, вращает взаимно перпендикулярные колесики. Благодаря наличию в них специальных прорезей оптическая система мыши способна отслеживать и преобразовывать движение шарика в перемещение курсора на экране компьютера. Две или три клавиши на верхней стороне мыши позволяют отдавать многочисленные команды, определяемые текущей программой. То же делает «перевернутая мышь» - шар (trackball), который монтируется в корпус компьютера или клавиатуры.

Есть еще ряд манипуляторов, служащих для ввода информации: **световое перо**, **джойстик** и т.д. Они второстепенны, решают некоторые ограниченные задачи.

Все чаще рядом с компьютером оказывается **сканер** - устройство для ввода с листа бумаги документов (текстов, чертежей и т.д.). Лучик света с огромной скоростью пробегает по листу, светочувствительными датчиками воспринимается яркость (а иногда и цветность) отраженного света

и трансформируется в двоичный код. Сканеры бывают цветными и монохромными, с разной разрешающей способностью, разным размером обрабатываемых изображений, настольными и ручными. Наиболее совершенные из них весьма дороги.

Несколько слов о вводе текстов с помощью сканера. Всякую информацию сканер воспринимает как графическую. Если это текст, то чтобы компьютер осознал его в таком качестве и позволил далее обрабатывать как текст (например, программами типа «редактор текстов»), нужна специальная программа распознавания, позволяющая выделить в считанном изображении отдельные символы и сопоставить им соответствующие коды символов. Это - достаточно сложная задача, но она успешно решается.

Не так давно появились *средства речевого ввода*, которые позволяют пользователю вместо клавиатуры, мыши и других устройств использовать речевые команды (или проговаривать текст, который должен быть занесен в память в виде текстового файла). Возможности таких устройств достаточно ограничены, хотя они постоянно совершенствуются. Проблема не в том, чтобы записать речь, подвергнуть ее дискретизации и ввести коды в компьютер (при современном уровне техники это несложно), а чтобы распознать смысл речи и представить ее, например, в текстовой форме, допускающей последующую компьютерную обработку. Например, программа «Kurzweil Voice 1.0 for Windows» обеспечивает распознавание (на английском языке) всех речевых команд для навигации в среде «Windows», а в режиме диктовки текста способна распознать до 40 тысяч слов, произносимых в среднем для человека темпе речи (требуя, однако, не менее 16 Мбайт ОЗУ и не менее 50 Мбайт на винчестере лишь для самой себя). Многие специалисты связывают с прогрессом устройств речевого ввода будущее компьютерной техники, считая такие устройства ведущими элементами ее интеллектуализации.

5.3. УСТРОЙСТВА ВЫВОДА ИНФОРМАЦИИ

Самым популярным из устройств вывода информации является *дисплей* - устройство визуального отображения текстовой и графической информации. Дисплей относится к числу неотъемлемых принадлежностей компьютера. Есть и параллельные термины, обозначающие почти то же самое: «видеотерминал», «видеомонитор» (хотя есть и смысловые оттенки: «монитор» - устройство управления чем-то, «терминал» - удаленное устройство доступа).

Дисплеи классифицируются по нескольким разным параметрам, отражающим их назначение в конкретной компьютерной системе и возможности. Бывают дисплеи монохромные и цветные. Монохромный дисплей производит отображение в двух цветах - черном и белом, либо зеленом и черном и т.д. Высококачественный цветной дисплей может воспроизводить десятки основных цветов и сотни оттенков.

Бывают дисплеи графические и алфавитно-цифровые (впрочем, последние, способные отображать лишь ограниченный набор основных символов используемого алфавита, почти исчезли из обычного обихода). Графический дисплей может отображать как символы, так и любое изображение, которое можно построить из отдельных точек в пределах разрешающей способности.

По физическим принципам, лежащим в основе конструкций дисплеев, подавляющее большинство их относится к дисплеям на базе электронно-лучевых трубок и к жидкокристаллическим дисплеям (последние особенно часто встречаются у портативных компьютеров). У первых формирование изображения производится на внутренней поверхности экрана, покрытого слоем люминофора - вещества, светящегося под воздействием электронного луча, генерируемого специальной «электронной пушкой» и управляемого системами горизонтальной и вертикальной развертки. Жидкокристаллический экран состоит из крошечных сегментов, заполненных специальным веществом, способным менять отражательную способность под воздействием очень слабого электрического поля, создаваемого электродами, подходящими к каждому сегменту.

При выводе на экран любого изображения, независимо от того, в растровом или векторном форматах оно зафиксировано в графических файлах, в видеопамяти формируется информация растрового типа, содержащая сведения о цвете каждого пиксела, задающего наиболее мелкую деталь изображения. Каждый пиксел однозначно связан с долей видеопамяти - несколькими битами, в которых программным путем задается яркость (и, при цветном экране, цветность) свечения этого пиксела. Специальная системная программа десятки раз в секунду считывает содержимое ви-

деопамяти и обновляет содержимое каждого пиксела, тем самым создавая и поддерживая на экране изображение. В п. 4.3 говорилось о том, как при этом взаимодействуют процессоры - центральный и контроллер дисплея.

Основные характеристики дисплеев с точки зрения пользователя таковы: разрешающая способность, число воспроизводимых цветов (для цветного дисплея) или оттенков яркости (для монохромного). Для алфавитно-цифрового дисплея разрешающая способность - число строк на экране и символов в каждой строке. Так, дисплей устаревшего отечественного компьютера ДБК-1 (диалоговый вычислительный комплекс) имел разрешающую способность 24x80 символов. Для графического - это число высвечиваемых точек по горизонтали и вертикали. К примеру, в табл. 4.4 приведены характеристики цветного графического дисплея SVGA (Super Video Graphics Adapter - видеографический адаптер повышенного разрешения) в нескольких из возможных режимах работы.

Таблица 4.4
Характеристики SVGA-монитора

Режим	Разрешающая способность	Число цветов (оттенков яркости)
Символьный	25x50	256
	51x132	256
Графический	768x1024	16
	480x640	64

В настоящее время (конец 90-х годов) начался промышленный выпуск плазменных дисплеев. В основе - возможность управлять возникновением электрических разрядов в некоторых газах и сопровождающим их свечением. Такие дисплеи обладают высоким качеством изображения и могут иметь значительно большие, чем у привычных компьютеров, размеры экранов при небольшой толщине (экран с диагональю около 1 м при толщине 8-10 см).

Огромную роль при выводе информации играют разнообразные печатающие устройства - **принтеры**. Наличие дисплея на современных компьютерах позволяет, работая в интерактивном режиме, экономить огромное количество бумаги, но все равно наступает, как правило, момент, когда необходима, так называемая, «твердая копия» информации - текст, данные, рисунок на бумаге. В процессе эволюции принтеры прошли следующий путь. Первые копировали пишущую машинку, имея ударные клавиши с буквами, цифрами и т.д. Под управлением процессора та или иная клавиша наносила удар по красящей ленте, оставляющей след на бумаге. Таких принтеров давно нет; их прямые наследники - точечно-матричные принтеры ударного типа - располагают перемещающейся вдоль строки печатающей головкой, содержащей от 7 до 24 игл, каждая из которых может независимо от остальных наносить удар по ленте. Это позволяет формировать изображения как букв и цифр, так и любых других символов, а также достаточно сложные рисунки и чертежи. Для хранения и подачи ленты используют специальную пластмассовую коробочку - картридж. Принтеры стали «интеллектуальными», т.е. имеют собственное ОЗУ и электронный блок управления для того, чтобы разгрузить основное ОЗУ и не отнимать в процессе печати время у центрального процессора.

Особенность современного принтера - возможность поддержки многих шрифтов. Часть шрифтов «прошита» в памяти принтера и задается нажатием клавиш на его панели. Еще больше шрифтов являются «загружаемыми», т.е. задаются той программой, которая обращается к устройствам печати. К примеру, широко распространенный графический редактор «Лексикон» позволяет пользователю выбирать между «обычным» шрифтом (так называемый шрифт пишущей машинки), жирным, курсивом (т.е. наклонными символами), жирным курсивом, символами с подчеркиванием и другими шрифтами - в зависимости от версии программы. Следует учесть, что при печати «собственными» шрифтами принтер обычно работает быстрее, так как комбинации ударов игл выбираются из знакогенератора принтера; загружаемые шрифты требуют дополнительного времени на загрузку до начала печати соответствующей программы - знакогенератора; самая медленная

печать осуществляется в графическом режиме, который требует постоянной пересылки в принтер информации о текущем режиме работы каждой иглы. Последний (графический) режим с появлением системы «Windows» стал очень распространенным; он не включает предварительной пересылки шрифтов в память принтера.

Приведем названия наиболее распространенных шрифтов, чаще всего «прошитых» в принтерах: roman - шрифт пишущей машинки; bold-face - полужирный; italic - курсив; condensed - сжатый.

Качество печати текста определяется не только шрифтом и классом принтера, но и числом точек, из которых формируется символ. Наиболее быстрый режим с минимально возможным числом точек и весьма невысоким качеством печати - режим черновой печати (draft); наиболее высококачественный - режим SLQ (Super Letter Quality). На одном и том же принтере соотношение скоростей печати в разных режимах может достигать 1:10.

Существуют ударные точечно-матричные принтеры цветной печати. В них используются 4-цветные ленты, и каждая точка изображения формируется четырьмя последовательными ударами иглы разной силы. Таким образом можно сформировать на бумаге точки всех основных цветов и множества оттенков. Крупнейший производитель точечно-матричных принтеров - фирма «Epson» (Япония).

Все чаще на рабочих местах пользователей персональных компьютеров появляются вместо точечно-матричных струйные или лазерные принтеры. Струйные принтеры вместо головки с иглами имеют головку со специальной краской и микросоплом, через которую эта краска «выстреливается» струйкой на бумагу (и быстро сохнет). Для формирования изображения либо струйка краски может отклоняться специально созданным электрическим полем (так как она электризуется в момент выхода из сопла), либо (чаще) головка имеет столбец из нескольких сопел - наподобие матрицы игл точечно-матричного принтера.

Струйные принтеры могут быть цветными, они смешивают на бумаге красители, порознь распыляемые разными соплами. Изображение, формируемое струйными принтерами, по качеству превосходит аналогичное, получаемое на точечно-матричных. Дополнительное достоинство - меньший уровень шума при работе.

Самые высококачественные изображения на бумаге на сегодняшний день дают лазерные принтеры. Один из основных узлов лазерного принтера - вращающийся барабан, на внешней поверхности которого нанесен специальный светочувствительный материал. Управляемый электронным блоком луч лазера оставляет на поверхности барабана наэлектризованную «картинку», соответствующую формируемому изображению. Затем на барабан наносится специальный мелкодисперсный порошок - тонер, частички которого прилипают к наэлектризованным участкам поверхности. Вслед за этим к барабану прижимается лист бумаги, на который переходит тонер, после чего изображение на бумаге фиксируется («прижигается») в результате прохождения через горячие валки. Все это происходит с огромной быстротой, благодаря чему лазерные принтеры значительно превосходят обсуждавшиеся выше по скорости работы. Лазерные принтеры - рекордсмены по части количества воспроизводимых шрифтов и качеству рисунков благодаря высочайшей разрешающей способности. Существуют как черно-белые, так и цветные лазерные принтеры. Лазерный принтер работает почти бесшумно. Единственный, но, увы, очень важный параметр, по которому они существенно уступают принтерам ранее описанных типов - стоимость; далеко не всякий может себе позволить приобрести принтер, по стоимости превосходящий точечно-матричный аналог в несколько раз.

Лидирующая фирма в производстве струйных и лазерных принтеров - «Hewlett-Packard» (HP), США, хотя в этой области действуют и другие фирмы.

Трудно сравнивать разные принтеры между собой, так как существует множество моделей с очень различающимися характеристиками. Приведем для примера принтеры разных типов примерно одного функционального класса, предназначенные для малого офиса или дома. Девятиигольчатый точечно-матричный EPSON LX 1050 при печати текста имеет максимальную производительность 200 знаков/с (при использовании встроенных шрифтов в режиме draft), т.е. может отпечатать страницу стандартного текста (порядка 30 строк) с очень скромным качеством за 10 с; высококачественная же печать займет до 2 мин. Струйный HP DeskJet 600 печатает со скоростью около 4 стр/мин с разрешением 600 точек/дюйм (весьма высокое качество печати); по скорости он превосходит указанный матричный примерно в 1,5 раза. Лазерный HP «LaserJet SL» по приведен-

ным характеристикам такой же, как указанный выше струйный, но по стоимости превосходит его примерно вдвое.

Существуют и принтеры, работающие на других физических принципах, но по распространенности они значительно уступают тем, которые обсуждались выше.

К принтерам близки по назначению **плоттеры** - специализированные устройства для вывода на бумагу чертежей и рисунков. Рисунок выполняется специальным пером, управляемым электронным блоком; для цветного плоттера необходимо несколько перьев. Плоттер необходим как часть АРМа проектировщика, инженера-конструктора, архитектора. В силу специализированности и высокой стоимости плоттеры не являются устройствами массового распространения.

Своеобразные устройства вывода - **синтезаторы звука**. Простейшие из них есть в арсенале почти у всех персональных компьютеров и представляют собой обычный малогабаритный динамик, напряжение сигнала на котором с большой частотой изменяется компьютером. Таким способом удастся подать простой звуковой сигнал, указывающий на наступление какого-либо события. Многие языки программирования дополняются командами типа BEEP, SOUND, позволяющими программировать серии звуков. Если звукогенератор физически реализован так, что частота звучания поддается регулированию, то можно запрограммировать несложную мелодию, а если есть несколько независимых звукогенераторов, то - и звучание оркестра. Для этого в современных компьютерах устанавливается специальная плата - звуковая карта, - способная преобразовывать аналоговый звуковой сигнал в последовательность двоичных цифр и наоборот. Существуют и синтезаторы речи, назначение которых понятно из названия.

Контрольные вопросы и задания

1. Перечислите основные устройства ввода и вывода информации.
2. На каких физических принципах работают различные устройства ввода/вывода?
3. Назовите основные профессиональные характеристики различных устройств ввода и вывода информации и их типичные значения.

§ 6. ЛОГИЧЕСКИЕ ОСНОВЫ ФУНКЦИОНИРОВАНИЯ ЭВМ

6.1. ЛОГИКА ВЫСКАЗЫВАНИЙ. ЭЛЕМЕНТАРНЫЕ ЛОГИЧЕСКИЕ ФУНКЦИИ

Люди, далекие от техники, часто смотрят на ЭВМ и другие цифровые электронные устройства как на нечто таинственное и непостижимое. Тем не менее, все эти устройства работают в строгом соответствии с четкими логическими законами. Знание и понимание этих законов помогает в общении с компьютером.

Для иллюстрации рассмотрим пример, описанный в одной из указанных ниже книг. Некий незадачливый бизнесмен подал в компьютерную фирму жалобу на приобретенный компьютер. По словам бизнесмена, компьютер неверно отвечал на поставленные вопросы. Прибывший для проверки жалобы специалист предложил бизнесмену продемонстрировать один из ошибочных ответов. Тот немедленно затребовал вывести список клиентов, проживающих в штатах Теннесси и Кентукки, на что компьютер бесстрастно объявил, что таковых не имеется. «Вот видите!» - кипятился бизнесмен. - «А я точно знаю, что и в том, и в другом штате есть множество клиентов!» Консультант попытался объяснить, что машина по-своему права и что человек не может жить одновременно в обоих штатах, но бизнесмен не хотел ничего слушать. Тогда консультант просто повторил запрос, заменив единственное слово (список клиентов, проживающих в штатах Теннесси ИЛИ Кентукки), и через полминуты вручил жалобщику распечатку требуемого списка. «А еще проще на первых порах делать два более простых запроса» - посоветовал он напоследок.

Главной причиной возникшего курьеза послужило незнание трех основных логических операций, лежащих в основе всех выводов компьютера. Иногда эти операции И, ИЛИ, НЕ называют «тремя китами машинной логики». Познакомимся с ними подробнее.

При записи тех или иных логических выражений используется специальный язык, который принят в математической логике. Основоположителем математической логики является великий немецкий математик Готфрид Вильгельм Лейбниц (1646 - 1716 гг.). Он сделал попытку построить универсальный язык, с помощью которого споры между людьми можно было бы разрешать посредством вычислений. На заложенном Лейбницем фундаменте ирландский математик Джордж

Буль построил здание новой науки - математической логики, - которая в отличие от обычной алгебры оперирует не числами, а высказываниями. В честь Д.Буля логические переменные в языке программирования Паскаль впоследствии назвали булевыми.

Высказывание - это любое утверждение, относительно которого можно сказать истинно оно или ложно, т.е. соответствует оно действительности или нет. Таким образом по своей сути высказывания фактически являются двоичными объектами и поэтому часто истинному значению высказывания ставят в соответствие 1, а ложному - 0. Например, запись $A = 1$ означает, что высказывание A истинно.

Высказывания могут быть простыми и сложными. Простые соответствуют алгебраическим переменным, а сложные являются аналогом алгебраических функций. Функции могут получаться путем объединения переменных с помощью логических действий.

Самой простой логической операцией является операция НЕ (по-другому ее часто называют отрицанием, дополнением или инверсией и обозначают NOT X). Результат отрицания всегда противоположен значению аргумента.

Логическая операция НЕ является унарной, т.е. имеет всего один операнд. В отличие от нее, операции И (AND) и ИЛИ (OR) являются бинарными, так как представляют собой результаты действий над двумя логическими величинами.

Таблица 4.5
Основные логические операции

X	NOT X
0	1
1	0

X	Y	X AND Y	X OR Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Логическое И еще часто называют конъюнкцией, или логическим умножением (не правда ли, таблица для этой операции похожа как две капли воды на двоичную таблицу умножения?), а ИЛИ - дизъюнкцией, или логическим сложением.

Операция И имеет результат «истина» только в том случае, если оба ее операнда истинны. Например, рассмотрим высказывание «Для остановки ОС «Windows'95» требуется процессор не ниже 80386 и не менее 4 Мбайт оперативной памяти». Из него следует, что установка будет успешной только при одновременном выполнении обоих условий: даже если у вас в машине Pentium, но мало ОЗУ (равно как и при 8 Мбайт ОЗУ процессор 80286), «Windows'95» работать откажется.

Операция ИЛИ «менее привередлива» к исходным данным. Она дает «истину», если значение «истина» имеет хотя бы один из операндов. Разумеется, в случае, когда справедливы оба аргумента одновременно, результат по-прежнему истинный. Действительно, когда студентка просит друга подарить ей на день рождения букет цветов или пригласить в кафе, можно без опасения сделать и то, и другое одновременно (впрочем, на практике в таком случае можно ограничиться чем-то одним).

Приведенные выше табл. 4.5 значений переменных для логических операций называются таблицами истинности. В них указываются все возможные комбинации логических переменных X и Y , а также соответствующие им результаты операций. Таблица истинности может рассматриваться в качестве одного из способов задания логической функции.

Операции И, ИЛИ, НЕ образуют полную систему логических операций, из которой можно построить сколь угодно сложное логическое выражение.

В вычислительной технике также часто используется операция исключающее ИЛИ (XOR), которая отличается от обыкновенного ИЛИ только при $X=1$ и $Y=1$.

Как видно из табл. 4.6, операция XOR фактически сравнивает на совпадение два двоичных разряда. Хотя теоретически основными базовыми логическими операциями всегда называют именно И, ИЛИ, НЕ, на практике по технологическим причинам в качестве основного логического элемента используется элемент И-НЕ (последняя колонка в табл. 4.6).

Таблица 4.6
Дополнительные логические операции

X	Y	X XOR Y	NOT(X AND Y)
0	0	0	1
0	1	1	1
1	0	1	1
1	1	0	0

Можно проверить, что на базе элементов И-НЕ могут быть сконструированы все базовые логические элементы (И, ИЛИ, НЕ), а значит и любые другие, более сложные.

6.2. СХЕМНАЯ РЕАЛИЗАЦИЯ ЭЛЕМЕНТАРНЫХ ЛОГИЧЕСКИХ ОПЕРАЦИЙ. ТИПОВЫЕ ЛОГИЧЕСКИЕ УЗЛЫ

Мы уже знаем, что любую достаточно сложную логическую функцию можно реализовать, имея относительно простой набор базовых логических операций. Первоначально этот тезис был технически реализован «один к одному»: были разработаны и выпускались микросхемы, соответствующие основным логическим действиям. Потребитель, комбинируя имеющиеся в его распоряжении элементы, мог получить схему с реализацией необходимой логики. Довольно быстро стало ясно, что подобное «строительство здания из отдельных кирпичиков» не может удовлетворить практические потребности. Промышленность увеличила степень интеграции МС и начала выпускать более сложные типовые узлы: триггеры, регистры, счетчики, дешифраторы, сумматоры и т.д. (продолжая аналогию со строительством, этот шаг, видимо, следует уподобить панельному способу домостроения). Новые микросхемы давали возможность реализовывать еще более сложные электронные логические устройства, но человеку свойственно не останавливаться на достигнутом: рост возможностей порождает новые потребности. Последовал переход к большим интегральным схемам (БИС), представлявшим из себя функционально законченные узлы, а не отдельные компоненты для их создания (как тут не вспомнить блочный метод постройки здания из готовых комнат). Наконец, дальнейшая эволюция технологий производства ИМС привела к настолько высокой степени интеграции, что в одной БИС содержалось функционально законченное изделие: часы, калькулятор, небольшая специализированная ЭВМ...

Если посмотреть на внутреннее устройство типичного современного компьютера, то там присутствуют ИМС очень высокого уровня интеграции: микропроцессор, модули ОЗУ, контроллеры внешних устройств и др. Фактически каждая микросхема или небольшая группа микросхем образуют функционально законченный блок. Уровень сложности блока таков, что разобраться в его внутреннем устройстве для неспециалиста не только нецелесообразно, а просто невозможно. К счастью, для понимания внутренних принципов работы современной ЭВМ достаточно рассмотреть несколько типовых узлов, а изучение поведения БИС заменить изучением функциональной схемы компьютера.

Обработка информации в ЭВМ происходит, как уже не раз отмечалось выше, путем последовательного выполнения элементарных операций. Эти операции менее многочисленны, нежели набор команд ЭВМ (которые реализуются через цепочки этих операций). К элементарным операциям относятся: установка - запись в операционный элемент (например, регистр) двоичного кода; прием - передача (перезапись) кода из одного элемента в другой; сдвиг - изменение положения кода относительно исходного; преобразование - перекодирование; сложение - арифметическое сложение целых двоичных чисел - и некоторые другие. Для выполнения каждой из этих операций сконструированы электронные узлы, являющиеся основными узлами цифровых вычислительных машин - регистры, счетчики, сумматоры, преобразователи кодов и т.д.

В основе каждой из элементарных операций лежит некоторая последовательность логических действий, описанных в предыдущем параграфе. Проанализируем, например, операцию сложения двух чисел: $3+6$. Имеем:

$$\begin{array}{r} + 110 \\ 1011 \end{array}$$

На каждом элементарнейшем шаге этой деятельности двум двоичным цифрам сопоставляется двоичное число (одно- или двузначное) по правилам: $(0,0) \Rightarrow 0$, $(0,1) \Rightarrow 1$, $(1,0) \Rightarrow 1$, $(1,1) \Rightarrow 10$. Таким образом, сложение цифр можно описать логической бинарной функцией. Если дополнить это логическим правилом переноса единицы в старший разряд (оно будет сформулировано ниже при описании работы сумматора), то сложение полностью сведется к цепочке логических операций.

Для дальнейшего рассмотрения необходимо знать условные обозначения базовых логических элементов. Они приведены на рис. 4.21. Соответствующие таблицы истинности приведены в предыдущем пункте.

Отметим, что на практике логические элементы могут иметь не один или два, а значительно большее число входов.

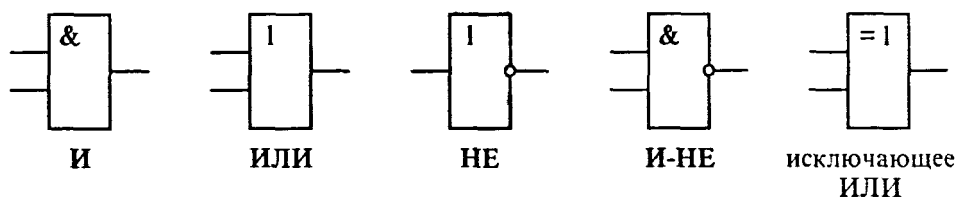


Рис. 4.21. Условные обозначения основных логических элементов

Итак, примем к сведению, что простейшие логические элементы, изображенные на рис. 4.21, можно реализовать аппаратно. Это означает, что можно создать электронные устройства на транзисторах, резисторах и т.п., каждое из которых имеет один или два входа для подачи управляющих напряжений и один выход, напряжение на котором определяется соответствующей таблицей истинности. На практике логическому «да» («истина», или цифра 1 в таблицах истинности) соответствует наличие напряжения, логическому «нет» («ложь», или цифра 0) - его отсутствие.

Вопрос, на который мы должны ответить, таков: как с помощью таких элементарных схем реализовать сложные цифровые устройства, необходимые для работы ЭВМ? При этом, учитывая существование прямых соответствий между логическими и электронными схемами, вполне достаточно достичь понимания на уровне логических схем.

В качестве характерных устройств выберем два наиболее важных и интересных - триггер (рис. 4.22) и сумматор. Первый - основа устройств оперативного хранения информации, второй служит для сложения чисел.

Перейдем к описанию работы триггера. Соответствующая его работе таблица истинности (табл. 4.7) приведена ниже.

Как видно из рис. 4.22, простейший вариант триггера собирается из четырех логических элементов И-НЕ, причем два из них играют вспомогательную роль. Триггер имеет два входа, обозначенные на схеме R и S, а также два выхода, помеченные буквой Q - прямой и инверсный (черта над Q у инверсного выхода означает отрицание). Триггер устроен таким образом, что на прямом и инверсном выходах сигналы всегда противоположны.

Как же работает триггер? Пусть на входе R установлена 1, а на S - 0. Логические элементы D1 и D2 инвертируют эти сигналы, т.е. меняют их значения на противоположные. В результате на вход элемента D3 поступает 1, а на D4 - 0. Поскольку на одном из входов D4 есть 0, независимо от состояния другого входа на его выходе (он же является инверсным выходом триггера!) обязательно установится 1. Эта единица передается на вход элемента D3 и в сочетании с 1 на другом входе порождает на выходе D3 логический 0. Итак, при $R=1$ и $S=0$ на прямом выходе триггера устанавливается 0, а на инверсном - 1.

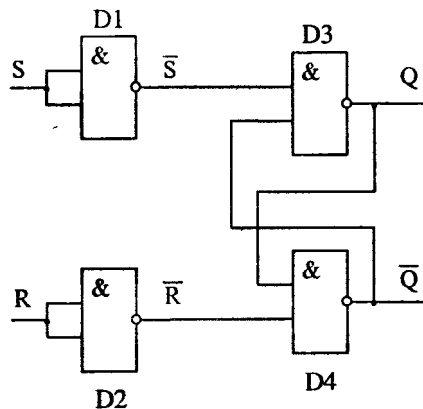


Рис. 4.22. Логическая схема триггера

Таблица 4.7
Таблица истинности RS-триггера

S	R	$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$	Примечания
0	0	1	1	X	X	Хранение
0	1	1	0	0	1	
1	0	0	1	1	0	
1	1	0	0	(1	1)	Запрещено

Обозначение состояния триггера по договоренности связывается с прямым выходом. Тогда при описанной выше комбинации входных сигналов результирующее состояние можно условно назвать нулевым: говорят, что триггер «устанавливается в 0» или «сбрасывается». Сброс по-английски называется «Reset», отсюда вход, появление сигнала на котором приводит к сбросу триггера, обычно обозначают буквой R.

Проведите аналогичные рассуждения для «симметричного» случая $R = 0$ и $S = 1$. Вы увидите, что на прямом выходе получится логическая 1, а на инверсном - 0. Триггер перейдет в единичное состояние - «установится» (установка по-английски - «Set»).

Теперь рассмотрим наиболее распространенную и интересную ситуацию $R = 0$ и $S = 0$ - входных сигналов нет. Тогда на входы элементов D3 и D4, связанные с R и S будет подана и их выходной сигнал будет зависеть от сигналов на противоположных входах. Нетрудно убедиться, что такое состояние будет устойчивым. Пусть, например, на прямом выходе 1. Тогда наличие единиц на обоих входах элемента D4 «подтверждает» нулевой сигнал на его выходе. В свою очередь, наличие 0 на инверсном выходе передается на D3 и поддерживает его выходное единичное состояние. Аналогично доказывается устойчивость картины и для противоположного состояния триггера, когда $Q = 0$.

Таким образом, при отсутствии входных сигналов триггер сохраняет свое «предыдущее» состояние. Иными словами, если на вход R подать 1, а затем убрать, триггер установится в нулевое состояние и будет его сохранять, пока не поступит сигнал на другой вход S. В последнем случае он перебросится в единичное состояние и после прекращения действия входного сигнала будет сохранять на прямом выходе 1. Мы видим, что триггер обладает замечательным свойством: после снятия входных сигналов он сохраняет свое состояние, а значит может служить устройством для хранения одного бита информации.

В заключение проанализируем последнюю комбинацию входных сигналов: $R = 1$ и $S = 1$. Нетрудно убедиться (проделайте необходимые рассуждения самостоятельно), что в этом случае на обоих выходах триггера установится 1! Такое состояние помимо своей логической абсурдности еще и является неустойчивым: после снятия входных сигналов триггер случайным образом перейдет в одно из своих устойчивых состояний. Вследствие этого, комбинация $R = 1$ и $S = 1$ никогда не используется на практике и является запрещенной.

Мы рассмотрели простейший RS-триггер. Существуют и другие разновидности этого инте-

ресного и полезного устройства. Все они различаются не столько принципом работы, сколько входной логикой, усложняющей «поведение» триггера.

Триггеры очень широко применяются в вычислительной технике. На их основе изготавливаются всевозможные регистры для хранения и некоторых видов обработки (например, сдвига) двоичной информации, счетчики импульсов и даже интегральные микросхемы статического ОЗУ, не требующие для сохранения информации специальных процессов регенерации. Множество триггеров входят в состав любого микропроцессора.

В качестве второго примера применения логических элементов в вычислительной технике рассмотрим устройство, называемое **сумматором**. Его назначение состоит в нахождении суммы двух двоичных чисел. Этот узел интересен для нас тем, что он лежит в основе арифметического устройства ЭВМ и иллюстрирует некоторые принципы выполнения вычислительных операций в компьютере.

Для простоты начнем с изучения логической структуры простейшего возможного устройства, являющегося звеном сумматора. Это устройство - **полусумматор** - реализует сложение двух одноразрядных двоичных чисел, которые обозначим А и В. В результате получается, вообще говоря, двухразрядное двоичное число. Его младшую цифру обозначим S, а старшую, которая при сложении многоразрядных чисел будет перенесена в старший разряд, через Co (от английских слов «Carry out»- «выходной перенос»). Для лучшего понимания происходящего вспомните правило типа «ноль пишем, один в уме».

Обе цифры можно получить по следующим логическим формулам:

$$S = (\bar{A} \wedge B) \vee (A \wedge \bar{B}) \quad C_0 = A \wedge B$$

(черта над символом обозначает операцию NOT, знак $\wedge$ - конъюнкцию, знак $\vee$ - дизъюнкцию). Это легко проверить перебором всех четырех возможных случаев сочетания значений А и В, пользуясь табл. 4.5 и табл. 4.8.

Таблица 4.8
Таблица истинности для полусумматора

A	B	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Мысленно объединим в табл. 4.8 столбцы А, В и Co. Полученная таблица напоминает базовый логический элемент И. Аналогично, сравнив первые три столбца А,В и S с имеющимися в предыдущем разделе таблицами истинности для распространенных логических элементов, обнаружим подходящий для наших целей элемент «исключающее ИЛИ». Таким образом, для реализации полусумматора достаточно соединить параллельно входы двух логических элементов (рис. 4.23).

Ниже приведены два варианта логической схемы полусумматора: с использованием лишь базовых логических элементов и с использованием логического элемента «исключающее ИЛИ». Видно, что вторая схема существенно проще.

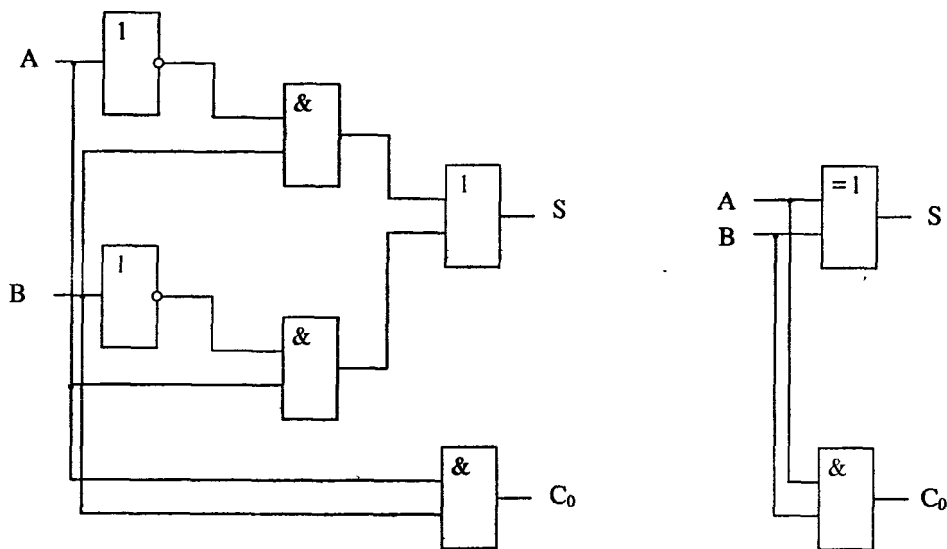


Рис. 4.23. Логическая схема полусумматора (два варианта)

Полный одноразрядный сумматор «умеет» при сложении двух цифр учитывать возможное наличие единицы, переносимой из старшего разряда (той, которая при обычном сложении столбиком остается «в уме»). Обозначим этот «бит переноса» через C_i (от английского «Carry in» - «входной перенос»).

Таблица 4.9
Таблица истинности для полусумматора

Входы			Выходы	
A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	1	1	0
0	1	0	0	1
1	0	1	1	0
1	0	0	0	1
1	1	0	0	1
1	1	1	1	1

При построении схемы сумматор удобно представить в виде двух полусумматоров, из которых первый суммирует разряды A и B, а второй к полученному результату прибавляет бит переноса C_i .

Заметим, что для суммирования младших разрядов чисел полусумматора уже достаточно, так как в этом случае отсутствует сигнал входного переноса.

Соединив два полусумматора как показано на рис. 4.24, получим полный сумматор, способный осуществить сложение двух двоичных разрядов с учетом возможности переноса.

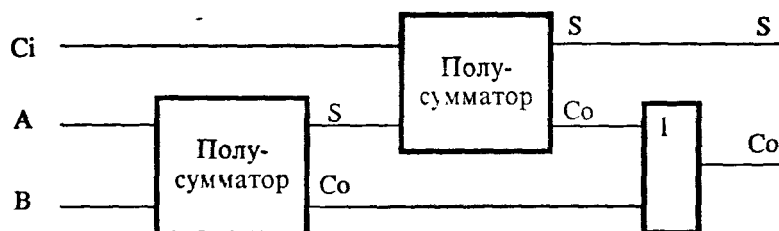


Рис. 4.24. Сумматор, составленный из двух полусумматоров

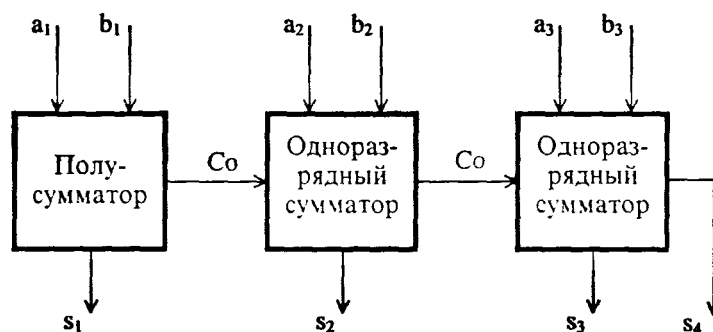


Рис. 4.25. Логическая схема суммирования двух трехразрядных двоичных чисел

Перейти к многоразрядным числам можно путем последовательного соединения соответствующего количества сумматоров. На рис. 4.25 представлена схема суммирования двух трехразрядных двоичных чисел $A + B = S$; в поразрядной записи эта операция имеет следующие обозначения:

$$(a_3a_2a_1) + (b_3b_2b_1) = (S_4S_3S_2S_1)$$

Последовательность логических схем на рис. 4.23 - 4.25 отражает важнейшую в современной цифровой электронике и вычислительной технике идею последовательной интеграции. Такая интеграция позволяет реализовать все более функционально сложные узлы современного компьютера.

63. ПРИМЕР ЭЛЕКТРОННОЙ РЕАЛИЗАЦИИ ЛОГИЧЕСКОГО ЭЛЕМЕНТА

Наконец, приведем обещанную выше электронную схему одного из элементарных логических устройств. Эта схема реализует логическую операцию И-НЕ.

Основа схемы - два рпн-транзистора. Для ее понимания достаточно вспомнить, как работают транзисторы. Через такой транзистор может протекать ток от коллектора к эмиттеру (на рис. 4.26 это соответствует «от плюса к минусу») при наличии положительного напряжения на базе (т.е. в точках А и В). Отсутствие напряжения на базе запирает этот ток.

Схема 4.26 имеет два входа А и В, через которые подается информация в виде электрического напряжения: есть напряжение - логическое «да», нет его - «нет». Выход - точка Y, наличие разности потенциалов между которой и точкой Z рассматривается как «да», отсутствие - как «нет». Питательное напряжение для схемы подается на левые входы («+» и «-»). Резистор R, при наличии тока, создает падение напряжения.

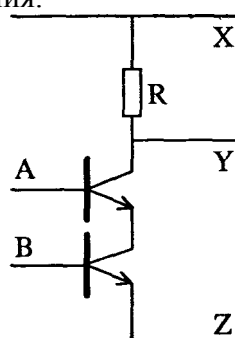


Рис. 4.26. Электронная реализация логического элемента И-НЕ (схема на рпн-транзисторах)

Допустим, на входах А и В нет напряжения («нет» и «нет»). В последней колонке табл. 4.6 этому соответствует $A = 0$, $B = 0$. Тогда оба транзистора «заперты», ток по цепи не протекает и между точками Y и Z есть разность потенциалов - т.е. результат операции «да», что в логических обозначениях соответствует 1 (именно как в табл. 4.6). Если заперт один из транзисторов, то результат все равно такой же (сравните с табл. 4.6). Лишь если оба транзистора открыты, ток в цепи идет и между точками Y и Z разности потенциалов нет (падение напряжения на самих транзисторах ничтожно мало по сравнению с его падением на резисторе).

За более подробными сведениями на эту тему следует обратиться к рекомендованным ниже

Контрольные вопросы и задания

1. Дайте определение понятию «высказывание» и приведите несколько примеров. Не забудьте объяснить, какие из них истинны, а какие ложны.
2. Самостоятельно (не заглядывая в книгу) постройте таблицы истинности для всех упоминающихся в тексте логических операций.
3. Вспомните, что вам известно об операциях И, ИЛИ, НЕ в языках программирования. Сопоставьте эти знания с только что построенными вами таблицами.
4. Нарисуйте условные обозначения основных логических элементов и постройте рядом их таблицу истинности.
5. Проанализируйте самостоятельно работу триггера и сумматора для каждой строчки таблицы истинности.
6. Попробуйте нарисовать схему соединения элементов И-НЕ для получения базовых логических операций. Если это не удастся, обратитесь за помощью к указанным ниже книгам.

Дополнительная литература к главе 4

1. *Александриди Т.М.* и др. Работа на микро-ЭВМ. - М.: Финансы и статистика, 1989.
2. *Апокин И. А., Майстров Л.Е.* История вычислительной техники. - М.: Наука, 1980.
3. *Богумирский Б. С.* Руководство пользователя ПЭВМ. Часть 1. Санкт-Петербург: Печатный Двор, 1994.
4. *Брусенцов Н.П.* Микрокомпьютеры. - М.: Наука, 1985.
5. *Василевский А. М., Кропоткин М.А., Тихонов В. В.* Оптическая электроника. -Л.: Энергоатомиздат, 1993.
6. *Васильев Б.М., Частиков А.П.* Микропроцессоры (история, применение, технология) //Информатика и образование. - 1993. -N 5. -С. 71-83.
7. *Гершензон Е. М., Полянина Г. Д., Соина Н. В.* Радиотехника. - М.: Просвещение, 1989.
8. *Гутер Р. С., Полунов Ю. Л.* От абака до компьютера. - М.: Знание, 1975.
9. *Джордейн Р.* Справочник программиста персональных компьютеров типа IBM PC, XT и AT. - М.: Финансы и статистика, 1992.
10. *Еремин Е.А.* Компилятор? Это очень просто. Компьютер УКНЦ. - М.: Компьютика, 1995.-N3.-С. 25-33.
11. *Еремин Е.А., Пономарева Л. В.* Методические рекомендации по изучению основных принципов работы ПК. - Пермь: ПГПУ, 1989.
12. *Еремин Е.А.* Как работает современный компьютер. - Пермь: из-во ПРИПИТ, 1997.
13. *Зальцман Ю.А.* Архитектура и программирование на языке ассемблера БК-0010.//Информатика и образование. 1990.-N4-6: 1991.-N1-5.
14. *Крук Б. И., Попов Г.Н.* ...И мир загадочный за занавесом цифр... цифровая связь. - М.: Радио и связь, 1992.
15. *Лин В.* PDP-11 и VAX-11. Архитектура ЭВМ и программирование на языке ассемблера. - М.: Радио и связь, 1989.
16. *Майоров С.А., Кириллов В.В., Приблуда А.А.* Введение в микро-ЭВМ. - Л.: Машиностроение, 1988.
17. *Манаев Е.И.* Основы радиоэлектроники. - М.: Радио и связь, 1990.
18. *Мнеян М.Г.* Физические принципы работы ЭВМ. - М.: Просвещение, 1987.
19. *Нортон П., Соухэ Д.* Язык ассемблера для IBM PC. - М.: Финансы и статистика, 1992.
20. *Нортон П.* Персональный компьютер фирмы IBM и операционная система MS DOS. М.: Радио и связь, 1992.
21. *Перегудов М.А., Халамайзер А. Я.* Бок о бок с компьютером. - М.: Высшая школа, 1987.
22. *Поляков В. Т.* Посвящение в радиоэлектронику. - М.: Радио и связь, 1990.
23. *Смит Б.Э., Джонсон М.Т.* Архитектура и программирование процессора INTEL 80386. - М.: Конкорд. 1992.
24. *Токхейм Р.* Основы цифровой электроники. - М.: Мир, 1988.
25. Толковый словарь по вычислительным системам / Под ред. В.Иллингворта и др. - М.:

Машиностроение, 1989.

26. *Частиков А. П.* История компьютера. - М.: Журнал «Информатика и образование», 1996.

27. *Эндерлайн Р.* Микроэлектроника для всех. - М.: Мир, 1979.

28. *Ямпольский В. С.* Основы автоматики и электронно-вычислительной техники. -М.: Просвещение, 1991.

ГЛАВА 5

КОМПЬЮТЕРНЫЕ СЕТИ И ТЕЛЕКОММУНИКАЦИИ

ВВЕДЕНИЕ

Компьютерная сеть - объединение нескольких ЭВМ для совместного решения информационных, вычислительных, учебных и других задач.

Одна из первых возникших при развитии вычислительной техники задач, потребовавшая создания сети хотя бы из двух ЭВМ - обеспечение многократно большей, чем могла дать в то время одна машина, надежности при управлении ответственным процессом в режиме реального времени. Так, при запуске космического аппарата необходимые темпы реакции на внешние события превосходят возможности человека, и выход из строя управляющего компьютера грозит непоправимыми последствиями. В простейшей схеме работу этого компьютера дублирует второй такой же, и при сбое активной машины содержимое ее процессора и ОЗУ очень быстро перебрасывается на вторую, которая подхватывает управление (в реальных системах все, конечно, происходит существенно сложнее).

Вот примеры других, очень разнородных, ситуаций, в которых объединение нескольких ЭВМ необходимо.

А. В простейшем, самом дешевом учебном компьютерном классе, лишь одна из ЭВМ - рабочее место преподавателя - имеет дисковод, позволяющий сохранять на диске программы и данные всего класса, и принтер, с помощью которого можно распечатывать тексты. Для обмена информацией между рабочим местом преподавателя и рабочими местами учеников нужна сеть.

Б. Для продажи железнодорожных или авиационных билетов, в которой одновременно участвуют сотни кассиров по всей стране, нужна сеть, связывающая сотни ЭВМ и выносных терминалов на пунктах продажи билетов.

В. Сегодня существует множество компьютерных баз и банков данных по самым разным аспектам человеческой деятельности. Для доступа к хранимой в них информации нужна компьютерная сеть.

Сети ЭВМ врываются в жизнь людей - как в профессиональную деятельность, так и в быт - самым неожиданным и массовым образом. Знания о сетях и навыки работы в них становятся необходимыми множеству людей.

Сети ЭВМ породили существенно новые технологии обработки информации -сетевые технологии. В простейшем случае сетевые технологии позволяют совместно использовать ресурсы - накопители большой емкости, печатающие устройства, доступ в Internet, базы и банки данных. Наиболее современные и перспективные подходы к сетям связаны с использованием коллективного разделения труда при совместной работе с информацией - разработке различных документов и проектов, управлении учреждением или предприятием и т.д.

Простейшим видом сети является, так называемая, одноранговая сеть, обеспечивающая связь персональных компьютеров конечных пользователей и позволяющая совместно использовать дисководы, принтеры, файлы.

Более развитые сети помимо компьютеров конечных пользователей - рабочих станций - включают специальные выделенные компьютеры - серверы. Сервер - это ЭВМ, выполняющая в сети особые функции обслуживания остальных компьютеров сети - рабочих станций. Есть разные виды серверов: файловые, телекоммуникационные серверы, серверы для проведения математических расчетов, серверы баз данных.

Весьма популярная сегодня и чрезвычайно перспективная технология обработки информа-

ции в сети называется «клиент - сервер». В методологии «клиент - сервер» предполагается глубокое разделение функций компьютеров в сети. При этом в функции «клиента» (под которым понимается ЭВМ с соответствующим программным обеспечением) входит

- предоставление пользовательского интерфейса, ориентированного на определенные производственные обязанности и полномочия пользователя;

- формирование запросов к серверу, причем не обязательно информируя об этом пользователя; в идеале пользователь вообще не вникает в технологию общения ЭВМ, за которой он работает, с сервером;

- анализ ответов сервера на запросы и предъявление их пользователю. Основная функция сервера - выполнение специфических действий по запросам клиента (например, решение сложной математической задачи, поиск данных в базе, соединение клиента с другим клиентом и т.д.); при этом сам сервер не инициирует никаких взаимодействий с клиентом. Если сервер, к которому обратился клиент, не в состоянии решить задачу из-за нехватки ресурсов, то в идеале он сам находит другой, более мощный, сервер и передает задачу ему, становясь, в свою очередь, клиентом, но не информируя об этом без нужды начального клиента. Обратим внимание, что «клиент» вовсе не есть выносной терминал сервера. Клиентом может быть весьма мощный компьютер, который в силу своих возможностей решает задачи самостоятельно.

Компьютерные сети и сетевые технологии обработки информации стали основой для построения современных информационных систем. Компьютер ныне следует рассматривать не как отдельное устройство обработки, а как «окно» в компьютерные сети, средство коммуникаций с сетевыми ресурсами и другими пользователями сетей.

§ 1. ЛОКАЛЬНЫЕ СЕТИ

1.1. АППАРАТНЫЕ СРЕДСТВА

Локальные сети (ЛС ЭВМ) объединяют относительно небольшое число компьютеров (обычно от 10 до 100, хотя изредка встречаются и гораздо большие) в пределах одного помещения (учебный компьютерный класс), здания или учреждения (например, университета). Традиционное название - локальная вычислительная сеть (ЛВС) - скорее дань тем временам, когда сети в основном использовались для решения вычислительных задач; сегодня же в 99% случаев речь идет исключительно об обмене информацией в виде текстов, графических и видео-образов, числовых массивов. Полезность ЛС объясняется тем, что от 60% до 90% необходимой учреждению информации циркулирует внутри него, не нуждаясь в выходе наружу.

Большое влияние на развитие ЛС оказало создание автоматизированных систем управления предприятиями (АСУ). АСУ включают несколько автоматизированных рабочих мест (АРМ), измерительных комплексов, пунктов управления. Другое важнейшее поле деятельности, в котором ЛС доказали свою эффективность - создание классов учебной вычислительной техники (КУВТ).

Благодаря относительно небольшим длинам линий связи (как правило, не более 300 метров), по ЛС можно передавать информацию в цифровом виде с высокой скоростью передачи. На больших расстояниях такой способ передачи неприемлем из-за неизбежного затухания высокочастотных сигналов, в этих случаях приходится прибегать к дополнительным техническим (цифро-аналоговым преобразованиям) и программным (протоколам коррекции ошибок и др.) решениям.

Характерная особенность ЛС - наличие связывающего всех абонентов высокоскоростного канала связи для передачи информации в цифровом виде. Существуют проводные и беспроводные (радио) каналы. Каждый из них характеризуется определенными значениями существенных с точки зрения организации ЛС параметров:

- скорости передачи данных;
- максимальной длины линии;
- помехозащищенности;
- механической прочности;
- удобства и простоты монтажа;
- стоимости.

В настоящее время обычно применяют четыре типа сетевых кабелей:

- коаксиальный кабель;

- незащищенная витая пара;
- защищенная витая пара;
- волоконно-оптический кабель.

Первые три типа кабелей передают электрический сигнал по медным проводникам. Волоконно-оптические кабели передают свет по стеклянному волокну.

Большинство сетей допускает несколько вариантов кабельных соединений.

Коаксиальные кабели состоят из двух проводников, окруженных изолирующими слоями. Первый слой изоляции окружает центральный медный провод. Этот слой оплетен снаружи внешним экранирующим проводником. Наиболее распространенными коаксиальными кабелями являются толстый и тонкий кабели «Ethernet». Такая конструкция обеспечивает хорошую помехозащищенность и малое затухание сигнала на расстояниях.

Различают толстый (около 10 мм в диаметре) и тонкий (около 4 мм) коаксиальные кабели. Обладая преимуществами по помехозащищенности, прочности, длине линий, толстый коаксиальный кабель дороже и сложнее в монтаже (его сложнее протягивать по кабельным каналам), чем тонкий. До последнего времени тонкий коаксиальный кабель представлял собой разумный компромисс между основными параметрами линий связи ЛВС и в российских условиях наиболее часто использовался для организации крупных ЛС предприятий и учреждений. Однако более дорогие толстые кабели обеспечивают лучшую передачу данных на большее расстояние и менее чувствительны к электромагнитным помехам.

Витые пары представляют собой два провода, скрученных вместе шестью оборотами на дюйм для обеспечения защиты от электромагнитных помех и согласования импеданса или электрического сопротивления. Другим наименованием, обычно употребляемым для такого провода, является «IBM тип-3». В США такие кабели прокладываются при постройке зданий для обеспечения телефонной связи. Однако использование телефонного провода, особенно когда он уже размещен в здании, может создать большие проблемы. Во-первых, незащищенные витые пары чувствительны к электромагнитным помехам, например электрическим шумам, создаваемым люминесцентными светильниками и движущимися лифтами. Помехи могут создавать также сигналы, передаваемые по замкнутому контуру в телефонных линиях, проходящих вдоль кабеля локальной сети. Кроме того, витые пары плохого качества могут иметь переменное число витков на дюйм, что искажает расчетное электрическое сопротивление.

Важно также заметить, что телефонные провода не всегда проложены по прямой линии. Кабель, соединяющий два рядом расположенных помещения, может на самом деле обойти половину здания. Недооценка длины кабеля в этом случае может привести к тому, что фактически она превысит максимально допустимую длину.

Защищенные витые пары схожи с незащищенными, за исключением того, что они используют более толстые провода и защищены от внешнего воздействия слоем изолятора. Наиболее распространенный тип такого кабеля, применяемого в локальных сетях, «IBM тип-1» представляет собой защищенный кабель с двумя витыми парами непрерывного провода. В новых зданиях лучшим вариантом может быть кабель «тип-2», так как он включает помимо линии передачи данных четыре незащищенные пары непрерывного провода для передачи телефонных переговоров. Таким образом, «тип-2» позволяет использовать один кабель для передачи как телефонных переговоров, так и данных по локальной сети.

Защита и тщательное соблюдение числа повивов на дюйм делают защищенный кабель с витыми парами надежным альтернативным кабельным соединением. Однако эта надежность приводит к увеличению стоимости.

Волоконно-оптические кабели передают данные в виде световых импульсов по стеклянным «проводам». Большинство систем локальных сетей в настоящее время поддерживает волоконно-оптическое кабельное соединение. Волоконно-оптический кабель обладает существенными преимуществами по сравнению с любыми вариантами медного кабеля. Волоконно-оптические кабели обеспечивают наивысшую скорость передачи; они более надежны, так как не подвержены потерям информационных пакетов из-за электромагнитных помех. Оптический кабель очень тонок и гибок, что делает его транспортировку более удобной по сравнению с более тяжелым медным кабелем. Однако наиболее важно то, что только оптический кабель имеет достаточную пропускную способность, которая в будущем потребуется для более быстрых сетей.

Пока еще цена волоконно-оптического кабеля значительно выше медного. По сравнению с

медным кабелем монтаж оптического кабеля более трудоемок, поскольку концы его должны быть тщательно отполированы и выровнены для обеспечения надежного соединения. Однако ныне происходит переход на оптоволоконные линии, абсолютно неподверженные помехам и находящиеся вне конкуренции по пропускной способности. Стоимость таких линий неуклонно снижается, технологические трудности стыковки оптических волокон успешно преодолеваются.

Беспроводная связь на радиоволнах СВЧ диапазона может использоваться для организации сетей в пределах больших помещений типа ангаров или павильонов, там где использование обычных линий связи затруднено или нецелесообразно. Кроме того, беспроводные линии могут связывать удаленные сегменты локальных сетей на расстояниях 3 - 5 км (с антенной типа волновой канал) и 25 км (с направленной параболической антенной) при условии прямой видимости. Организация беспроводной сети существенно дороже, чем обычной.

Для организации учебных ЛС чаще всего используется витая пара, как самая дешевая, поскольку требования к скорости передачи данных и длине линий не являются критическими.

Для связи компьютеров с помощью линий связи ЛС требуются адаптеры сети (или, как их иногда называют, сетевые платы). Самыми известными являются адаптеры следующих трех типов: • ArcNet; • Token Ring; • Ethernet.

Из них последние получили в России подавляющее распространение. Адаптер сети вставляется непосредственно в свободный слот материнской платы персонального компьютера и к нему на задней панели системного блока подстыковывается линия связи ЛС. Адаптер, в зависимости от своего типа, реализует ту или иную стратегию доступа от одного компьютера к другому.

1.2. КОНФИГУРАЦИИ ЛОКАЛЬНЫХ СЕТЕЙ И ОРГАНИЗАЦИЯ ОБМЕНА ИНФОРМАЦИЕЙ

В простейших сетях с небольшим числом компьютеров они могут быть полностью равноправными; сеть в этом случае обеспечивает передачу данных от любого компьютера к любому другому для коллективной работы над информацией. Такая сеть называется одноранговой.

Однако в крупных сетях с большим числом компьютеров оказывается целесообразным выделять один (или несколько) мощных компьютеров для обслуживания потребностей сети (хранение и передачу данных, печать на сетевом принтере). Такие выделенные компьютеры называют серверами; они работают под управлением сетевой операционной системы. В качестве сервера обычно используется высокопроизводительный компьютер с большим ОЗУ и винчестером (или даже несколькими винчестерами) большой емкости. Клавиатура и дисплей для сервера сети не обязательны, поскольку они используются очень редко (для настройки сетевой ОС).

Все остальные компьютеры называются рабочими станциями. Рабочие станции могут не иметь винчестерских дисков или даже дисководов вовсе. Такие рабочие станции называют бездисковыми. Первичная загрузка ОС на бездисковые рабочие станции происходит по локальной сети с использованием специально устанавливаемых на сетевые адаптеры рабочих станций микросхем ПЗУ, хранящих программу начальной загрузки.

ЛС в зависимости от назначения и технических решений могут иметь различные конфигурации (или, как еще говорят, архитектуру, или топологию), изображенные на рис. 5.1.

В *кольцевой* ЛС информация передается по замкнутому каналу. Каждый абонент непосредственно связан с двумя ближайшими соседями, хотя в принципе способен связаться с любым абонентом сети.

В *звездообразной (радиальной)* ЛС в центре находится центральный управляющий компьютер, последовательно связывающийся с абонентами и связывающий их друг с другом.

В *шинной* конфигурации компьютеры подключены к общему для них каналу (шине), через который могут обмениваться сообщениями.

В *древовидной* - существует «главный» компьютер, которому подчинены компьютеры следующего уровня, и т.д.

Кроме того, возможны конфигурации без отчетливого характера связей; пределом является *полносвязная* конфигурация, когда каждый компьютер в сети непосредственно связан с любым другим компьютером.

В крупных ЛС предприятий и учреждений чаще всего используется шинная (линейная) топология, соответствующая архитектуре многих административных зданий, имеющих длинные ко-

ридоры и кабинеты сотрудников вдоль них. Для учебных целей в КУВТ чаще всего используют кольцевые и звездообразные ЛС.

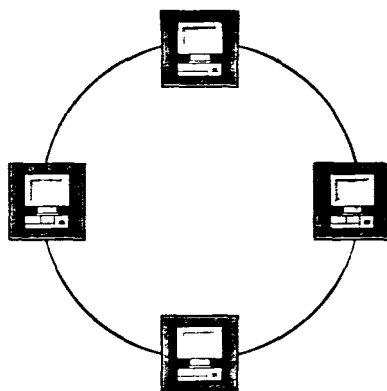


Рис. 5.1,0. Кольцевая топология - локальной сети

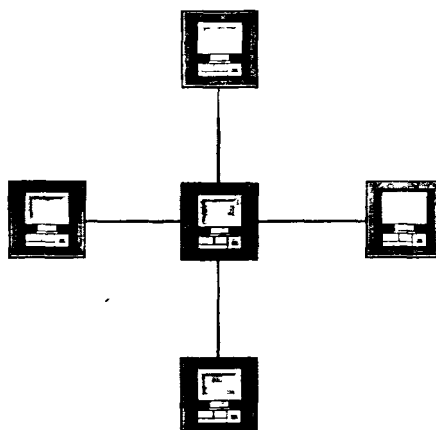


Рис. 5.1,6. Радиальная топология локальной сети

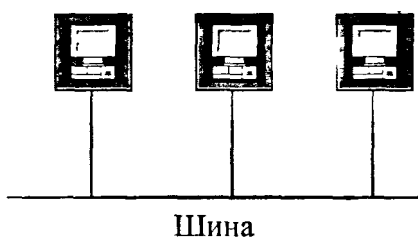


Рис. 5.1,в. Шинная топология локальной сети

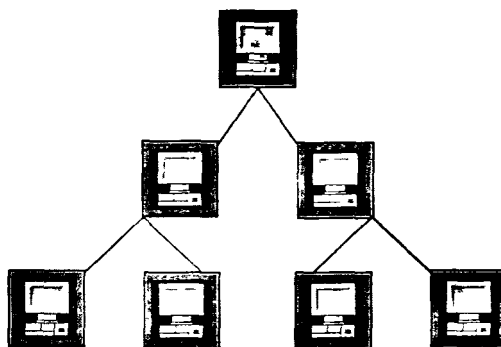


Рис. 5.1,г. Древовидная топология локальной сети

В любой физической конфигурации поддержка доступа от одного компьютера к другому, наличие или отсутствие выделенного компьютера (в составе КУВТ его называют «учительским», а остальные - «ученическими»), выполняется программой - сетевой операционной системой, кото-

рая по отношению к ОС отдельных компьютеров является надстройкой. Для современных высококоразвитых ОС персональных компьютеров вполне характерно наличие сетевых возможностей (например, OS/2, WINDOWS'95-98).

Процесс передачи данных по сети определяют шесть компонент:

- компьютер-источник;
- блок протокола;
- передатчик;
- физическая кабельная сеть;
- приемник;
- компьютер-адресат.

Компьютер-источник может быть рабочей станцией, файл-сервером, шлюзом или любым компьютером, подключенным к сети. Блок протокола состоит из набора микросхем и программного драйвера для платы сетевого интерфейса. Блок протокола отвечает за логику передачи по сети. Передатчик посылает электрический сигнал через физическую топологическую схему. Приемник распознает и принимает сигнал, передающийся по сети, и направляет его для преобразования в блок протокола.

Как показано на рис. 5.2, цикл передачи данных начинается с компьютера-источника, передающего исходные данные в блок протокола. Блок протокола организует данные в пакет передачи, содержащий соответствующий запрос к обслуживающим устройствам, информацию по обработке запроса (включая, если необходимо, адрес получателя) и исходные данные для передачи. Пакет затем направляется в передатчик для преобразования в сетевой сигнал. Пакет распространяется по сетевому кабелю пока не попадает в приемник, где перекодировается в данные. Здесь управление переходит к блоку протокола, который проверяет данные на сбойность, передает «квитанцию» о приеме пакета источнику, переформировывает пакеты и передает их в компьютер-адресат.

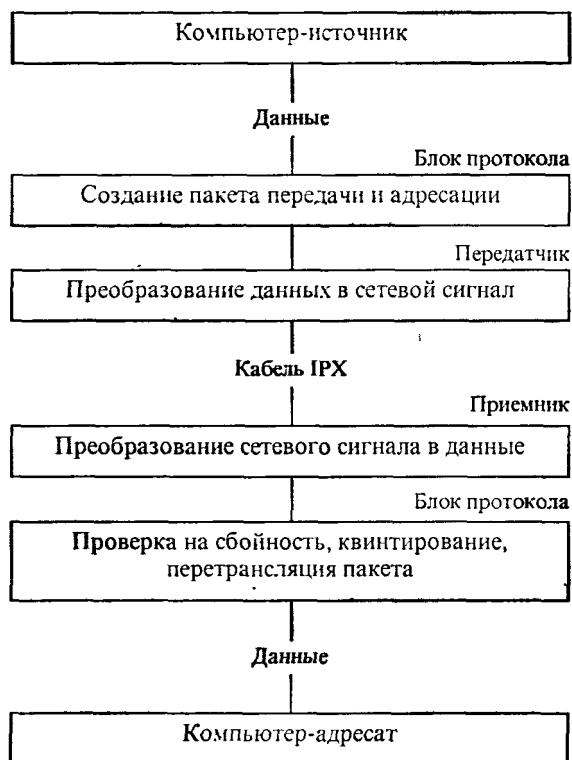


Рис. 5.2. Схема переноса информации в сети

В ходе процесса передачи блок протокола управляет логикой передачи по сети через схему доступа.

Каждая сетевая ОС использует определенную стратегию доступа от одного компьютера к другому. Широко используются маркерные методы доступа (называемые также селективной передачей), когда компьютер-абонент получает от центрального компьютера сети, так называемый, маркер - сигнал на право ведения передачи в течение определенного времени, после чего маркер передается другому абоненту. При конкурентном методе доступа абонент начинает передачу дан-

ных, если обнаруживает свободной линией, или откладывает передачу на некоторый промежуток времени, если линия занята другим абонентом. При другом способе - резервировании времени - у каждого абонента есть определенный промежуток, в течение которого линия принадлежит только ему.

Наиболее часто применяются две основные схемы:

- конкурентная (Ethernet);
- с маркерным доступом (Token Ring, Arcnet).

Ведутся дебаты о том, какая схема более эффективна - конкурентная или с маркерным доступом. Сети с маркерным доступом обычно более медленные, но обладают более предсказуемыми свойствами, чем конкурентные. По мере роста числа пользователей у сетей с маркерным доступом параметры ухудшаются медленнее, чем у конкурентных сетей. Эффективность сети зависит от величины потока сообщений, который необязательно связан с числом активных рабочих станций. При конкурентной схеме, когда много рабочих станций одновременно пытаются переслать данные, возникают наложения. Таким образом, если большая часть обработки данных в сети выполняется локально (например, если рабочие станции заняты, главным образом, локальной подготовкой текстов), эффективность сети будет высокой, даже если к сети подключено много пользователей.

При схеме с маркерным доступом эффективность непосредственно определяется числом активных рабочих станций, а не полным потоком сообщений, передаваемых по сети. Каждый дополнительный пользователь добавляет еще один адрес, по которому будет передан маркер независимо от того, нуждается или нет рабочая станция в пересылке сообщения.

Сеть Ethernet использует для управления передачей данных по сети конкурентную схему. Элементы сети Ethernet могут быть соединены по шинной или звездной топологии с использованием витых пар, коаксиальных или волоконно-оптических кабелей.

Основным преимуществом сетей Ethernet является их быстродействие. Обладая скоростью передачи от 10 до 100 Мбит/с, Ethernet является одной из самых быстрых среди существующих локальных сетей. Однако такое быстродействие, в свою очередь, вызывает определенные проблемы: из-за того, что предельные возможности тонкого медного кабеля лишь незначительно превышают указанную скорость передачи в 10 Мбит/с, даже небольшие электромагнитные помехи могут значительно ухудшить производительность сети.

Как показывает их наименование, сети **IBM Token Ring** используют для передачи данных схему с маркерным доступом. Сеть Token Ring физически выполнена по схеме «звезда», но ведет себя как кольцевая. Другими словами, пакеты данных передаются с одной рабочей станции на другую последовательно (как в кольцевой сети), но постоянно проходят через центральный компьютер (как в сетях типа «звезда»). Сети Token Ring могут осуществлять передачу как по незащищенным и защищенным витым проводным парам, так и по волоконно-оптическим кабелям.

Сети Token Ring существуют в двух версиях, со скоростью передачи в 4 или 16 Мбит/с. Однако, хотя отдельные сети работают на скоростях либо 4, либо 16 Мбит/с, возможно соединение через мосты сетей с разными скоростями передачи. Сети Token Ring надежны, обладают высокой скоростью (особенно версия со скоростью передачи 16 Мбит/с) и просты для установки. Однако по сравнению с сетями ARCnet сети Token Ring дороги.

Сеть **ARCnet** использует схему с маркерным доступом и может работать как в шинной, так и в звездной топологии. Схема «звезда» обычно обеспечивает лучшую производительность, так как при этой топологии возникает меньше конфликтов при передаче. ARCnet совместима с коаксиальными кабелями, витыми парами и волоконно-оптическими кабелями.

Системы ARCnet являются сравнительно медленными. Передача осуществляется на скорости лишь 2,5 Мбит/с, что значительно меньше, чем в других типах сетей. Несмотря на малое быстродействие, ARCnet сохраняет свою популярность. Ее малая скорость передачи является в своем роде компенсацией за эффективный метод передачи сигналов. ARCnet - сравнительно недорогая и гибкая система, которая легко устанавливается, расширяется и подвергается изменению конфигурации.

Правила организации передачи данных в сети называют протоколом. Определенный протокол поддерживается как аппаратно (адаптерами сети), так и программно (сетевой ОС).

В ЛС данные передаются от одного компьютера к другому блоками, которые называют пакетами данных. Станция, передающая пакет данных, обычно указывает в его заголовке адрес

назначения данных и свой собственный адрес. Пакеты могут передаваться между рабочими станциями без подтверждения - это тип связи на уровне датаграмм. Проверка правильности передачи пакетов в этом случае выполняется сетевой ОС, которая может сама посылать пакеты, подтверждающие правильную передачу данных. Важное преимущество датаграмм - возможность отправки пакетов сразу всем станциям в сети.

Например, протокол передачи данных IPX (от слов «Internetnetwork Packet Exchange», что означает «межсетевой обмен пакетами») используется в сетевом программном обеспечении фирмы «Novell» и является реализацией датаграмм. Другой пример - разработанный фирмой IBM протокол NETBIOS, также получивший большую известность, тоже работает на уровне датаграмм.

Сетевой адрес состоит из нескольких компонентов:

- номера сети;
- адреса станции в сети;
- идентификатора программы на рабочей станции.

Номер сети - это номер сегмента сети (кабельного хозяйства), определяемого системным администратором при установке сетевой ОС.

Адрес станции - это число, являющееся уникальным для каждой рабочей станции. Уникальность адресов при использовании адаптеров Ethernet обеспечивается заводом-изготовителем плат (адрес станции записывается в микросхеме ПЗУ адаптера). На адаптерах ArcNet адрес станции устанавливается при помощи перемычек или микропереключателей.

Идентификатор программы на рабочей станции называется **сокет**. Это - число, которое используется для адресации пакетов в конкретной программе, работающей на станции под управлением многозадачной операционной системы (типа Windows, OS/2). Каждая программа для того, чтобы посылать или получать данные по сети, должна получить свой, уникальный для данной рабочей станции, идентификатор - сокет.

1.3. ЛОКАЛЬНЫЕ СЕТИ УЧЕБНОГО НАЗНАЧЕНИЯ

ЛС КУВТ - совокупность аппаратных и программных средств, ориентированных на использование в учебном процессе. В нашей стране в конце 80-х годов получили широкое распространение КУВТ «Ямаха» (японского производства), отечественные КУВТ на базе микро-ЭВМ БК0010, УКНЦ, «Корвет». Им на смену пришли КУВТ на базе компьютеров IBM PC (и им подобных) и «Apple Macintosh». В ряде мест функционируют и гибридные КУВТ с головной машиной IBM PC и ученическими УКНЦ или «Корвет».

В состав каждого КУВТ входят:

- рабочее место преподавателя (РМП);
- рабочие места учащихся (РМУ) - обычно 10 - 15;
- аппаратные и программные средства сетеобразования. В составе РМП обязательно находится компьютер (системный блок, дисплей и клавиатура), достаточно емкое устройство для хранения информации – накопитель и принтер. В указанных выше КУВТ первого поколения обычно роль накопителя выполняли два НГМД и бытовой кассетный магнитофон. Разумеется, такая сеть предоставляет весьма слабые возможности; в современных ЛС КУВТ на головной машине находится винчестер с емкостью до 3 Гбайт, CD ROM, другие устройства.

Сетевая ОС, функционирующая на РМП, должна предоставлять следующий минимальный набор пользовательских возможностей:

- пересылку программ и данных с РМП на каждое из РМУ и обратно;
- исполнение программ как на РМУ, так и на РМП;
- вывод программ и данных с РМУ на внешние накопители и принтер РМП;
- групповую рассылку программ с РМП на все РМУ.

В ходе этой работы ОС ЛС КУВТ должна быть способной к следующему.

1. Поддержка файловой системы. Это связано с необходимостью обеспечить абонентам - учащимся - доступ к файлам, хранящимся на головной машине сети, которая в этом случае исполняет роль файлового сервера. В более «продвинутом» варианте на головной машине может иметься база данных, представляющая интерес для учебного процесса, и ОС должна поддерживать доступ к этой базе.

2. Защита данных и разграничение доступа. Без этого файлы одних учащихся при записи на

общий диск сотрут файлы других. Кроме того, в такой системе коллективного пользования могут быть конфиденциальные данные, и система должна предусмотреть вариант их защиты от несанкционированного доступа (например, по паролю).

3. Система контроля и ведения урока. Она включает возможность преподавателю вмешиваться в работу учащихся, просматривать их экраны, вызывать и редактировать их программы, организовывать коллективные демонстрации и т.д.

Высокоразвитые ОС ЛС КУВТ предоставляют немалые возможности. Среди команд преподавателя есть несколько справочных, позволяющих установить в каком режиме функционируют компьютеры учащихся, команды пересылки программ г. их автоматического запуска на РМУ, команды вызова файлов - программ и данных - с любого из РМУ на РМП или на диск, отключения любого из РМУ от сети и обратное подключение. Сеть поддерживает локальную электронную почту - обмен короткими текстовыми сообщениями между любыми компьютерами.

Очень важен такой показатель как быстродействие сети. Так, скорость передачи по исходной ЛС КУВТ УКНЦ в 5-8 кбит/с приводит, например, к затрате нескольких минут на рассылку компилятора Паскаля - это слишком много для учебного процесса. Установка в этом классе головной машины IBM PC с сетевой системой фирмы «Линтех» сокращает это время минимум в 10 раз. Однако, даже в классах на основе компьютеров IBM PC и Macintosh скорость рассылки по сети бывает недостаточно высокой, что создает проблемы при учебной работе.

В качестве конкретного примера приведем ЛС КУВТ «Ямаха», ориентированную на работу с Бейсик-системой. Хотя этот КУВТ и устарел, он все еще используется во многих педагогических вузах, а программное обеспечение его ЛС является примером удачной реализации ЛС учебного назначения. Сетевые возможности реализованы в операционной среде MSX-BASIC, загружающейся на все компьютеры по умолчанию при отсутствии в дисковом А системной дискеты.

Полный набор команд, которыми управляет учитель, равен 25. Не обделены и ученики, которым подвластны либо 13 (расширенный набор), либо 9 команд (малый набор) по усмотрению учителя. Подав команду DISCOM либо ENACOM со своего компьютера, учитель задает возможности учеников.

Каждый ученик может по собственному усмотрению вовсе отключиться от сети, подав команду NETINIT и, тем самым, выйти из под контроля, одновременно лишив себя возможности работы с диском и принтером. Командой ONLINE ученик вновь может подключиться к сети. Команда CHECK позволяет учителю узнать, какие компьютеры подключены к сети, а какие - нет (на экране учительской машины появляется ответ). Учитель может и сам отключить со своего места любой ученический компьютер от сети: SNDC (_"offline", n) (n = 1, 2, ... 15 - по числу рабочих мест в классе).

Пересылка программ с РМП на любое из РМУ реализуется командами SEND:

- SEND (<имя_программы>; n) - с диска РМП на РМУ;
- SEND (<имя_программы>; 0) - с диска РМП на все ученические машины одновременно;
- SEND (, n) - из ОЗУ РМП на РМУ;
- SEND (, 0) - из ОЗУ РМП на все ученические машины одновременно.

Добавив к слову SEND приставку RUN (т.е. подав команду SENDRUN), учитель может организовать немедленный запуск программы на исполнение на ученической машине.

Для пересылки файлов данных используется команда BSEND (аналогично команде SEND).

Учитель может получить программу с любого ученического компьютера командой RECEIVE (name, n) или RECEIVER n).

Если ученик располагает расширенным набором команд, то он тоже может «перетягивать» к себе на компьютер чужую программу: RECEIVE (,n) или RECEIVE (, 0) или RECEIVE (<имя_программы>,0).

В сети реализована возможность обмена текстовыми сообщениями между любыми компьютерами. Например, команда TALK ("здравствуй, Миша!", 12), введенная в любой ученический компьютер, выведет на экране компьютера N 12 указанный текст.

Контрольные вопросы

1. Что такое компьютерная сеть?
2. Для чего создаются локальные сети ЭВМ?
3. Что такое сервер? рабочая станция ?

4. Какие сетевые технологии называются клиент-серверными?
5. Что такое сетевой адаптер? Какие типы сетевых адаптеров существуют ?
6. Какие виды линий (каналов) используются для связи компьютеров в локальных сетях ?
7. Какие методы доступа от компьютера к компьютеру используются в локальных сетях?
8. Какие бывают конфигурации ЛС?
9. Какие конфигурации ЛС используются в компьютерных классах Вашего вуза?
10. Какая сетевая ОС используется в ЛС, в которой вы работаете?
11. Чем отличается набор команд этой ОС от описанного выше?
12. Каковы специфические функции локальной сети учебного назначения?

§2. ОПЕРАЦИОННЫЕ СИСТЕМЫ ЛОКАЛЬНЫХ СЕТЕЙ

Появление компьютерных сетей привело к развитию операционных систем для персональных компьютеров, позволяющих работать в сетях. Такие операционные системы обеспечивают не только совместное использование аппаратных ресурсов сети (принтеров, дисковых накопителей большой емкости и т.д.), но и использование распределенных коллективных технологий при выполнении разнообразных работ.

Существует много операционных систем локальных сетей. Наиболее широкое распространение получили операционные системы **Novell NetWare** и **Windows NT** для локальных сетей ПК. Ознакомимся с первой из них.

Фирма «Novell Inc.», в компьютерном мире не менее известная чем фирмы IBM и «Microsoft», специализируется на создании операционных систем локальных сетей. Созданная в 1982 г. небольшой группой менеджеров и программистов, фирма уже к 1990 г. имела годовой оборот на уровне 500 млн. долларов. Ее сетевые ОС известны своим высочайшим качеством и надежностью.

Сетевая операционная система **Novell NetWare 386**, начиная с версии 3.11, представляет собой 32-разрядную операционную систему реального времени, работающую в защищенном режиме процессоров 80386 и более мощных (80486 и **Pentium**).

NetWare 386 является сетевой ОС с централизованным управлением, т.е. в сети один или несколько компьютеров должны быть выделены в качестве файл-серверов. На файл-серверах работает ОС **NetWare 386**. Остальные компьютеры сети, число которых может достигать нескольких сотен, являются рабочими станциями, и на них должна быть загружена, так называемая, клиентская часть **NetWare 386** -специальная компонента системы.

ОС **Netware 386** имеет ядро - файл **server.exe**, который запускается из MS DOS (на файл-сервере), затем с консоли и с помощью текстового файла конфигурации загружаются разнообразные драйверы, управляющие сетевыми устройствами. Кроме драйверов могут быть запущены в многозадачном режиме разнообразные утилиты и прикладные программы, например, управляющие файлами, хранящимися на сервере, и базами данных, работающими с внешними коммуникационными каналами. Эти команды содержатся в файлах, имеющих имена с расширением **.nlm** и потому называются «nlm-модулями».

Примером важнейшей утилиты является **syscon.exe**, с помощью которой администратор системы выполняет всю работу по разграничению доступа пользователей к информации.

После установки **NetWare** на файл-сервере на его винчестерском накопителе обязательно создается системный том SYS, содержащий несколько стандартных директорий:

- LOGIN, содержащую программу подключения пользователя к сети **hgin.exe** и другие процедуры, обслуживающие потребности пользователя рабочей станции при подключении ее к сети;
- MAIL, в которой для каждого пользователя сети заводится отдельный подкаталог (с именем из цифр), содержащий стартовый файл пользователя **Login Script** и файл конфигураций заданий на печать;
- SYSTEM, в которой хранятся файлы операционной системы **NetWare**, системные утилиты и некоторые другие служебные программы; этот каталог виден только системному администратору (имеющему идентификатор SUPER VISOR);
- PUBLIC, в которой содержатся сетевые программы и утилиты, доступные для всех пользователей сети;
- USERS, с индивидуальными подкаталогами пользователей сети. Файловая система **NetWare** на сервере не совместима с MS DOS. На рабочих же станциях, после запуска клиентской

части системы, обеспечивается прозрачный доступ к разделам диска файл-сервера как к своим собственным дисководам.

Файловая система **NetWare** поддерживает разветвленную систему разграничения доступа к файлам и каталогам файл-сервера с различных рабочих станций. Все пользователи сети могут быть разделены системным администратором на группы. Каждая группа обладает своими правами доступа, притом один и тот же пользователь может находиться одновременно в разных группах. В табл. 5.1 приведены типичные для **Novell NetWare** виды доступа к каталогам и файлам.

Таблица 5.1
Некоторые виды доступа к каталогам и файлам

Виды доступа	Обозначение	Права
Access Control	A	Изменение прав доступа к каталогу и файлу
File Stream	F	Просмотр содержимого каталога
Create	C	Создание каталогов или файлов в данном каталоге
Erase	E	Удаление каталогов или файлов из данного каталога
Modify	M	Изменение содержимого файлов (перезапись)
Supervisory	S	Права супервизора (позволяют делать любые операции с файлами, расположенными в каталоге)
Write	W	Запись в файл

Определение групп и их прав доступа выполняется с помощью утилиты **syscon.exe**.

Операционная система обладает развитым командным языком для написания файлов загрузки **Login Script**, большим набором утилит для пользователя и системного администратора. Охарактеризуем важнейшие из них. Для пользователя это утилиты, хранящиеся в каталоге PUBLIC.

LOGIN - первая команда, которую должен выдать пользователь перед началом работы в сети. При помощи этой команды пользователь подключается к файл-серверу.

Если команда **login** запущена без параметров, вы будете подключены к ближайшему серверу. Дополнительно Вам надо будет ввести имя пользователя и, возможно, пароль:

`login [/параметры][сервер[/имя_пользователя]][параметры_входа]`

В табл. 5.2 приведены параметры, задаваемые командой **login** через символ "/" (выделенные буквы можно использовать в качестве сокращения).

Таблица 5.2
Некоторые параметры команды LOGIN

Параметр	Назначение и использование
/Script	Позволяет выполнить подключение к сети с отличным от стандартного файлом Login Script; после параметра необходимо указать полный путь к нужному файлу Login Script
/NoAttach	Этот параметр нужен для запуска файла Login Script без подключения к файл-серверу
/CleanScreen	Стирание экрана во время ввода пароля

После перечисленных выше параметров можно указать имя файл-сервера и имя пользователя. После имени пользователя можно указать дополнительные параметры, которые можно будет проанализировать во время выполнения файла **Login Script**.

LOGOUT - отключение от файл-сервера. В качестве параметра можно указать имя файл-сервера, от которого вы собираетесь отключиться:

`logout[имя_сервера]`

Если имя файл-сервера не указано, выполняется отключение от всех серверов, к которым

вы были подключены.

MAP - отображение каталогов файл-сервера на локальные диски рабочей станции. Эта команда позволяет автоматизировать процесс отображения каталогов (табл. 5.3). Приведем общий формат вызова команды (с некоторыми упрощениями):

map [функция][диск:=[путь[,...]]]

Таблица 5.3
Форматы команды MAP

Формат	Комментарий
map	Если команда map выдана без параметров, на экране отображается карта текущего отображения каталогов
map диск:= [путь] каталог	Отображение диска «диск» на сетевой каталог «каталог»; в пути для сетевого каталога можно указывать имя сервера или имя тома.
map диск:= [путь] каталог; диск:= [путь] каталог;	Одновременное отображение нескольких дисков в одной команде
map insert диск_поиска:= [путь] каталог;	Отображение сетевого каталога на диск. Полученный диск автоматически просматривается при запуске программ, поэтому он называется «дискпоиска»
map root диск := [путь] каталог; диск := диск	Отображение сетевого каталога на корневой каталог локального диска. Требуется для тех программ, которые могут работать только с корневым каталогом

MENU. Если, находясь в каталоге SYS:PUBLIC, вы запустите программу *menu.exe* с параметром **main**, на экране появится меню, с помощью которого можно выполнять запуск различных сетевых утилит (табл. 5.4).

NCOPY - копирование файлов (табл. 5.5). В отличие от средств копирования файлов, предоставляемых MS DOS, утилита *ncopy* сохраняет атрибуты файлов, специфические для **Novell NetWare**, поэтому она более предпочтительна для использования в сети, чем программы MS DOS:

ncopy [путь1]имя_файла [to] путь2 [параметры...]

В строке, приведенной выше, «путь1» и «путь2» определяют соответственно пути к копируемому файлу и к каталогу, в который он будет копироваться. Можно указывать символы "*", "?", которые будут восприниматься так же, как и для обычной команды сору из операционной системы MS DOS. Для переименования файла при копировании необходимо в качестве параметра «путь2» использовать полный путь к новому файлу с указанием нового имени.

NDIR - получение детальной информации о файлах, расположенных в сетевых каталогах:

ndir [путь][параметры...]

Для команды ndir можно задавать параметры, можно просмотреть полный список параметров, запустив утилиту ndir с параметром /HELP.

SALVAGE - восстановление случайно удаленных файлов. Восстановление возможно, если не использовалась утилита *purge*, очищающая каталоги от удаленных файлов.

Запустив утилиту, оказываемся в меню **Main Menu Options**. Выбрав строку **Select Current Directory**, указываем каталог, в котором необходимо восстановить удаленные файлы. Указанный каталог становится текущим (с точки зрения утилиты **salvage**).

Таблица 5.4
Меню для запуска сетевых утилит

Строка	Утилита	Назначение

Session Management	session	Управление сеансами связи
File Management	filer	Управление файловой системой
Volume Information	volinfo	Получение информации о сетевых томах
System Configuration	syscon	Утилита для изменения конфигурации системы
File Server Monitoring	fconsole	Управление файл-сервером
Print Queue Management	pconsole	Управление очередями печати
Print Job Configuration	printcon	Определение конфигурации заданий на печать
Printer Definitions	printdef	Определение принтеров
Logout	logout	Отключение от сети

Таблица 5.5
Параметры процедуры NCOPY

Параметр	Назначение и использование
/S	Копирование вместе с подкаталогами
/E	Копирование пустых подкаталогов. Этот ключ используется совместно с ключом /S
/C	Копирование файлов без сохранения атрибутов
/I	Выдать сообщением тех случаях, когда программа не может скопировать атрибуты файла
/V	Выполнить проверку после копирования на идентичность файла-оригинала и файла-копии
/A	Скопировать файлы с установленным в слове атрибутов архивным битом. Атрибуты файла не изменяются
/M	Скопировать файлы с установленным в слове атрибутов архивным битом. После копирования архивный бит в слове атрибутов исходного файла сбрасывается. Это позволяет использовать NCOPY для выполнения архивной выгрузки содержимого каталога

Для восстановления файлов необходимо выбрать в главном меню строку **Salvage From Deleted Directories**. Для просмотра списка удаленных файлов в текущем каталоге и их последующего восстановления служит строка **View/Recover Deleted Files**.

SEND - посылка с одной рабочей станции на другие короткого (длиной не более 44 символов) сообщения.

Если рабочая станция, на которую послано сообщение, находится в текстовом режиме, полученное сообщение отображается в нижней строке экрана. Работа рабочей станции приостанавливается до тех пор, пока пользователь не нажмет комбинацию клавиш <Ctrl+Enter>.

SESSION - просмотр информации о группах пользователей и об отдельных пользователях, посылка пользователям сообщения, просмотр и изменение отображения сетевых каталогов на диски рабочей станции.

SYSCON - управление сервером, группами пользователей и отдельными пользователями в диалоговом режиме.

Обычные пользователи с помощью этой утилиты могут изменить свой пароль и отредактировать свой файл **Login Script**. Пользователи с правами супервизора могут с помощью этой утилиты назначать руководителей групп.

FCONSOLE - управление файл-сервером. Эта диалоговая утилита способна выполнять такие операции с файл-сервером, которые обычно выполняются при помощи команд с консоли файл-сервера. С ее помощью можно выдавать сообщения одновременно всем пользователям в сети, просматривать информацию о работающих в сети пользователях и управлять состоянием файл-сервера.

FILER - управление томами и каталогами. С помощью этой утилиты можно изменять атрибуты каталогов и файлов, изменять права доступа пользователей к каталогам и файлам.

С консоли файл-сервера (если к нему подключена клавиатура и дисплей) можно вводить разнообразные команды. Наиболее часто употребляемые команды приведены ниже.

BROADCAST - посылка сообщения пользователям. Это могут быть, например, сообщения о завершении работы сервера или о размонтировании сетевых томов. Длина сообщения не может превышать 55 символов:

broadcast <сообщение> to [параметр]

DISMOUNT - размонтирование сетевого тома, после чего он делается недоступным для пользователей. Такая операция может потребоваться для снятия редко используемого тома или восстановления его утилитой **vrepair**:

dismount <имя_тома>

Перед размонтированием тома необходимо предупредить об этом пользователей при помощи команды **broadcast**.

DOWN - останов файл-сервера. При этом содержимое дисковых буферов переписывается из оперативной памяти на диск, закрываются все файлы и обновляются каталоги. Перед выдачей этой команды необходимо предупредить пользователей, послав им сообщение при помощи команды **broadcast**.

После завершения процесса останова операционной системы **Novell NetWare** версии 3.11 на экран консоли выдается сообщение:

Server ...has been shut down. Type EXIT to return to DOS

После появления этого сообщения необходимо выдать команду **exit**. Операционная система **Novell NetWare** завершит свою работу, и появится системное приглашение MS DOS. Если была выдана команда **remove dos**, вместо возврата в MS DOS произойдет «теплая» перегрузка файл-сервера (без выполнения процедур начального тестирования при включении питания).

EXIT - возвращение в MS DOS или теплая перегрузка файл-сервера, если была выдана команда **remove dos**. Команда **exit** должна выдаваться после команды **down**, останавливающей файл-сервер. Эта команда отсутствует в операционной системе **Novell NetWare** версии 2.2.

LOAD - запуск с консоли файл-сервера nlm-программ, для загрузки драйверов сетевых адаптеров и дисковых драйверов. Эта команда отсутствует в операционной системе **Novell NetWare** версии 2.2, так как в этой версии не используется механизм nlm-программ.

MODULES - просмотр списка программ, работающих в среде **Novell NetWare**, на экране консоли файл-сервера.

На экран выводится таблица, в которой для каждой программы приводится имя файла, содержащего программу, и краткое название программы. Эта команда отсутствует в операционной системе **Novell NetWare** версии 2.2. Список загруженных VAP-процессов можно просмотреть по команде VAP.

MOUNT- монтирование тома, имя которого задано в качестве параметра. Если в качестве параметра указать "ALL", будут смонтированы все имеющиеся на файл-сервере тома. Размонтировать том можно при помощи команды **dismount**.

SET - просмотр и установление различных параметров **Novell NetWare**. Детальное рассмотрение этих параметров выходит за рамки нашей книги; их можно найти в документации, поставляющейся с **Novell NetWare**.

SET TIME - установка системной даты:

set time [месяц/день/год][часы:минуты:секунды]

UNLOAD - выгрузка nlm-программ, запущенных при помощи команды **load**. Единственный параметр этой команды - имя выгружаемой программы. Эта команда отсутствует в операционной системе **Novell NetWare** версии 2.2.

VOLUMES - вывод на экран консоли файл-сервера списка томов, смонтированных на данном файл-сервере. Эта команда отсутствует в операционной системе **Novell NetWare** версии 2.2.

В систему **NetWare** версии 3.11 входит ряд полезных nlm-модулей (находящихся в каталоге SYSTEM). Они могут быть запущены с помощью команды **load**. Среди них:

EDIT - запуск с консоли файл-сервера текстового редактора, если ввести команду **load edit**. После загрузки в ответ на приглашение **File to Edit** необходимо ввести путь к файлу, который надо отредактировать, например:

sys:system/autoexec.ncf

Завершение редактирования - клавиша <ESC> и подтверждение необходимости сохранения файла (выбором строки **Yes** в появившемся меню).

MONITOR - отображение текущего состояния сети. С помощью этой команды можно заблокировать клавиатуру консоли файл-сервера паролем. Программа запускается командой **load monitor**.

VREPAIR - исправление небольших разрушений в логической структуре данных на сетевых томах. Загружается при помощи команды **load**.

Перед восстановлением тома его необходимо размонтировать при помощи команды **dis-mount**.

Контрольные вопросы и задания

1. Каково назначение и возможности сетевой ОС **Novell NetWare**?
2. Охарактеризуйте основные команды **NetWare**.
3. Охарактеризуйте основные утилиты **NetWare**.

§3. ГЛОБАЛЬНЫЕ СЕТИ

3.1. ОБЩИЕ ПРИНЦИПЫ ОРГАНИЗАЦИИ

Глобальные информационные сети - одно из основных достижений человечества в области информационных технологий, главная примета вхождения в эпоху информационного общества. Делая возможным оперативное общение на огромных расстояниях (в разных странах и даже на континентах), глобальные сети уже изменили для многих людей характер и возможности образования и профессиональной деятельности. Потенциальные возможности глобальных сетей пока используются лишь в малой мере, но эта область прикладной информатики является самой динамичной.

Пользователю глобальной сети доступен, по существу, весь мир. Самой известной глобальной сетью является **Internet**, представляющий собой объединение огромного числа сетей - национального, отраслевого и еще более узкого - регионального уровня. «Малые» сети имеют выходы (шлюзы) в сети более высокого ранга, в согласованную систему адресов и протоколов (правил) передачи данных, и так образуют **Internet** - сеть сетей.

В глобальных сетях существует два режима информационного обмена. Диалоговый режим (или режим реального времени), в котором пользователь, получив порцию информации, может немедленно на нее реагировать, подавать новую команду в сеть для получения новых порций информации, называется **on-line**. В пакетном режиме, называемом **off-line**, пользователь передает порцию информации (или принимает ее) в коротком сеансе связи и на некоторое время отключается от сети. Это время может быть достаточно длительным - от нескольких часов до нескольких суток - пока его запрос не будет обработан. **On-line** похож на разговор по телефону, **off-line** - на обмен обычными письмами по почте.

Первая сеть с коммутацией пакетов ARPANET начала действовать в США в 1969 г. Тогда в ней было всего лишь 4 компьютера. В настоящее время на Западе действует много глобальных сетей. Назовем для примера BITNET - сеть, объединяющую более 800 коллективных участников, преимущественно из числа университетов, колледжей и научных центров, охватывающую 35 стран Америки, Европы и Азии.

Из российских телекоммуникационных сетей крупнейшей (фактически действующей во всех странах СНГ и Прибалтики) является сеть RELCOM, созданная в 1990 г. Уже в 1996 г. она имела порядка 300 узлов и насчитывала десятки тысяч абонентов. Сеть является членом Европейского консорциума EUNET, объединяющего многие сети Восточной Европы и Северной Африки, которая, в свою очередь, является участником гигантского мирового сообщества INTERNET. Еще в 1994 г. пользователи RELCOM имели доступ к более чем миллиону компьютеров по всему миру. Шлюз, позволяющий RELCOM выходить в зарубежные сети, находится в Москве, отсюда информация, адресованная за рубеж, поступает в Хельсинки - Амстердам - и далее по всему миру с использованием всех существующих видов связи. Для оптимизации работы сети RELCOM она, имея в основе форму звезды, включает, тем не менее, множество перекрестных соединений между уз-

лами, что позволяет установить дублирующие каналы связи и резко увеличить надежность и «живучесть» сети. Сеть универсальна, она предоставляет все виды услуг - от простой электронной почты в режиме **off-line** до выхода в **Internet** в режиме **on-line**.

Подключившись к сети Relcom, пользователь получает возможность обмениваться любой информацией со всеми абонентами этой сети и десятков других сетей. Это может быть личная или служебная переписка, обмен научной и деловой информацией, двоичные данные (программы, графические файлы) и многое другое. При этом можно общаться с конкретным пользователем, с группой пользователей или со всеми пользователями, интересующимися данным вопросом (с помощью, так называемых, телеконференций). Компьютер пользователя с помощью модема соединяется по телефонной линии с ближайшим узлом сети RELCOM, забирает пришедшую для него информацию и передает новые сообщения. Узел сети анализирует адреса новых сообщений и, в зависимости от результата, передает информацию либо сразу на компьютер адресата, либо на ближайший к адресату узел. Узлы между собой могут быть соединены коммутируемыми линиями, выделенными линиями, оптоволоконным кабелем, спутниковой связью.

RELCOM является официально зарегистрированной национальной подсетью европейской сети EUnet, поэтому ее абоненты имеют возможность пользоваться соглашениями о взаимной передаче сообщений между сетью EUnet и другими сетями и системами электронной почты, охватывающими весь земной шар. При этом услуги сети могут оплачиваться в национальной валюте.

EUnet/RELCOM входит в мировую систему телеконференций USENET, и в ее рамках участвует более чем в полутора тысячах распространяемых по всему земному шару конференциях.

Кроме RELCOM'a в России действуют не менее трех десятков независимых сетей. Большинство из них являются ведомственными и имеют относительно немного клиентов. Например, любительская компьютерная сеть - **FidoNet**. В качестве каналов связи в ней используются практически только коммутируемые телефонные каналы. Пользователь сети имеет возможность обмена почтовыми сообщениями и файлами. На большинстве узлов сети действуют электронные «доски объявлений». Работа в сети ведется в режиме **on-line** только в определенные часы.

Электронные доски объявлений (BBS) предназначены для обмена файлами между пользователями. Они организуются на какой-либо ЭВМ (станции). Файлы (тексты, программы и пр.) на доску объявлений могут помещать как владельцы станции, так и все остальные пользователи. Большинство существующих досок объявлений для связи с удаленными пользователями используют обычные телефонные линии.

Абонент связывается с доской объявлений любым доступным ему способом и за свой счет. При ответе со стороны станции он работает с ней в режиме удаленного терминала как один из пользователей, но с ограниченными правами.

Доска объявлений предоставляет следующие виды услуг:

- просмотреть содержание (список файлов);
- скопировать файл с доски объявлений на свою ЭВМ;
- передать файл на доску объявлений;
- некоторые станции организуют для своих абонентов ящики электронной почты и предоставляют услуги в рамках только этой станции или одной из систем электронной почты.

Для того, чтобы сеть могла функционировать, кроме абонентских пунктов в ней существуют специальные компьютерные узлы связи, функционирующие круглосуточно (их головные компьютеры называют хост-машинами). Если же они, как часто бывает, наряду с управлением сетью хранят банки информации, то их называют серверами (последний термин используется в литературе в нескольких смыслах в зависимости от контекста).

Конечный пользователь включается в обмен по своему усмотрению в желаемое для него время. Он соединяется с ближайшей к нему хост-машиной, которая, в свою очередь, соединяется с другими хост-машинами в соответствии с адресом, указываемым пользователем сети, затем происходит обмен информацией. Если пользователь работает в режиме **on-line**, то он сразу получает поступающую информацию и подает новые команды в сеть для получения новых порций информации. В режиме **off-line** пользователь сети при соединении с хост-машиной лишь передает свои сообщения и получает поступившую для него к этому времени информацию. Хост-машина хранит поступившие на нее сообщения до тех пор, пока пользователь их не заберет (или ограничивается разумным сроком хранения). Хост-машины постоянно связаны между собой по выделенным каналам связи и обмениваются информацией в автоматическом режиме под управлением специальных

программ. Структура такой сети схематически изображена на рис. 5.3.

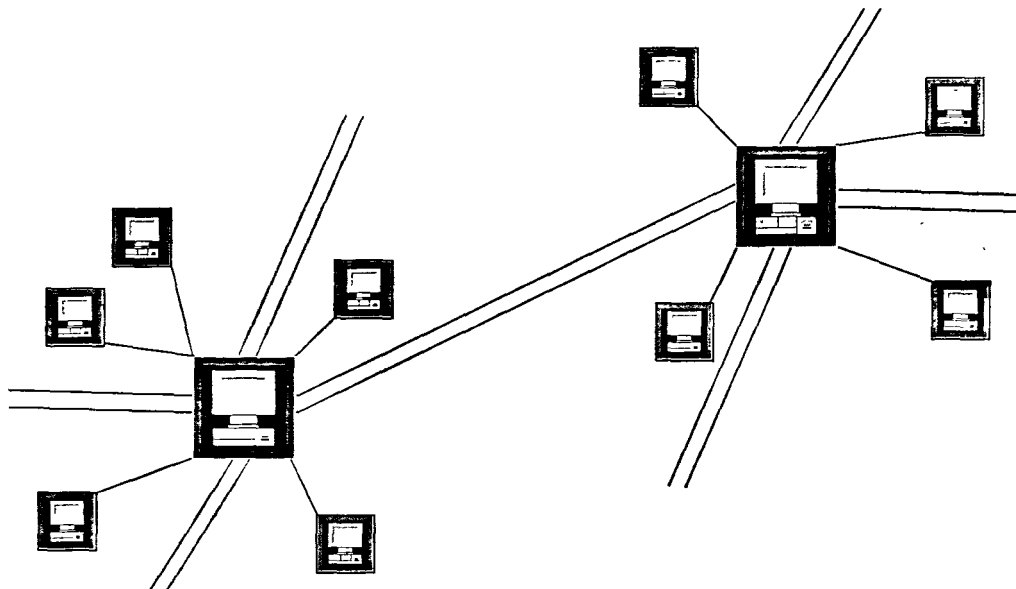


Рис. 5.3. Структура глобальной компьютерной сети

3.2. АППАРАТНЫЕ СРЕДСТВА И ПРОТОКОЛЫ ОБМЕНА ИНФОРМАЦИЕЙ

Основное техническое устройство сети - компьютер. Подавляющая часть компьютеров в сети являются абонентскими пунктами, т.е. обслуживают конечных пользователей - потребителей информации. В этом качестве компьютер выполняет следующие функции: хранение и обработку информации, подготовку ее к передаче, печать (при необходимости) на бумажном носителе, управление процессом передачи информации.

Транспортной основой глобальных сетей выступают коммутируемые и выделенные телефонные линии и каналы (как обычные, медные, так и оптоволоконные и спутниковые). Соединения в коммутируемых линиях происходят через стандартное оборудование телефонных станций (коммутаторы) при наборе телефонных номеров. Выделенные каналы (они соединены постоянно) используются, например, при организации сетей с **on-line** сервисом, особенно для соединения серверов, находящихся в разных городах.

Непременным оборудованием абонентского пункта является модем - устройство преобразования цифровых сигналов в аналоговые и наоборот. Преобразование это необходимо (по крайней мере, в настоящее время) при использовании телефонных линий связи общего назначения, приспособленных к передаче аналоговых сигналов звуковой частоты (т.е. модулированного электрического тока).

Основные характеристики модемов:

- максимальная скорость передачи данных;
- поддерживаемый сетевой протокол;
- протокол коррекции ошибок.

Модем по своим входным и выходным характеристикам должен соответствовать требованиям к оборудованию, разрешенному для подключения к телефонной сети. Работа модема определяется системой его команд и протоколом, стандартизованным

Международным консультативным комитетом по телефонии и телеграфии (МККТТ). Ныне наиболее широко используются, так называемые, Hayes-совместимые модемы (по фамилии разработчика системы команд Дениза Хейза) с протоколами, поддерживающими скорость передачи данных от 1200 бит/с до 54000 бит/с.

Модемы выпускаются в двух конструктивных исполнениях: встроенные и внешние. Конструктивное исполнение никак не влияет на качество работы модема, однако внешний модем более удобен в эксплуатации.

Большинство высокоскоростных модемов совместимы с менее быстрыми. Для повышения скорости и надежности обмена информацией используются, так называемые, MNP-модемы - модемы с аппаратным сжатием и коррекцией информации.

По видам передаваемой информации бывают следующие разновидности модемов:

- только для обмена данными между компьютерами;
- факс-модемы (передача данных + прием-передача факсов);
- факс-голос-модемы.

По способу «общения» различают *дуплексный* и *полудуплексный* режимы передачи данных. В дуплексном режиме данные передаются через модем одновременно в обоих направлениях. Для ряда применений возможен лишь этот режим. По способу группирования данных различают асинхронную (позначную) и синхронную (поблочную) передачу. Не вдаваясь в детали, отметим, что синхронная передача организуется сложнее, но обеспечивает более высокую эффективность.

MNP-модемы различаются по классам. Каждый класс отличается от предыдущего более высокой производительностью и расширенными возможностями.

Класс 1 использует асинхронный полудуплексный метод передачи данных с побайтной организацией. Сравнительная эффективность составляет 70%, т.е. MNP-модем класса 1, работающий со скоростью 2400 бит/с, передает полезную информацию со скоростью 1680 бит/с.

Класс 2 использует асинхронный дуплексный метод передачи данных с побайтной организацией. Сравнительная эффективность 84%.

Класс 3 использует синхронный дуплексный метод передачи данных с побитной организацией. Сравнительная эффективность 108%.

Класс 4 отличается тем, что в нем реализованы два новых метода работы с информацией: адаптивная сборка передаваемых блоков и оптимизация фазы. Сравнительная эффективность 120%.

Класс 5 использует в дополнение к возможностям класса 4 сжатие данных в реальном масштабе времени. Коэффициент сжатия может достигать 90% для некоторых видов информации. Графические файлы могут сжиматься до 10% исходного размера, текстовые - до 45-55%, программы - до 60-90%. Средний коэффициент сжатия 63%. Относительная эффективность 200%.

Класс 6 в дополнение к возможностям класса 5 обеспечивает совместимость высокоскоростных протоколов с низкоскоростными стандартами.

Класс 7 использует более совершенный алгоритм сжатия данных. Сравнительная эффективность 300%.

Класс 8 применяет более совершенный протокол и соответствующий метод работы, обеспечивающий совместимость с низкоскоростными модемами.

Казалось бы, чем выше скорость передачи, тем лучше модем. Однако чтобы сохранить хорошее качество линий передачи, приходится вводить ограничения на скорость, при которой число сбоев на линии не выходит за приемлемое (модем - интеллектуальное устройство, которое отслеживает сбои и либо повторяет передачу информации небольшими блоками до успешного конца, либо прерывает ее). В российских городских сетях приемлемая скорость передачи невелика и составляет 2400-14400 бит/с (максимальная скорость модемной связи на момент написания этой книги равна 54000 бит/с). В будущем цифровая связь может сделать ненужной аналого-цифровые преобразователи и модемы (по крайней мере, в их нынешнем виде).

Поскольку сеть объединяет компьютеры различных типов, работающие в разнородных ОС, важнейшим моментом при организации сети является система протоколов. **Протокол** - совокупность правил, согласно которым компьютеры взаимодействуют между собой. Различные компьютеры сети могут использовать совершенно разные программные средства, лишь бы пересылаемые данные соответствовали правилам протоколов. Международный стандарт OSI/ISO (OSI - Open Systems Interconnect, ISO - название международной организации по стандартизации) предусматривает 7 уровней протоколов, в числе которых отметим протоколы сетевого уровня, обеспечивающие сетевые режимы передачи данных (самый распространенный из них называется X.25), транспортные протоколы, отвечающие за обмены между разными хост-машинами сети. При этом обмен ведется чаще всего пакетами, т.е. группами сообщений. Прикладные протоколы обслуживают задачи пользователя по передаче данных и доступу к сетевым ресурсам. Межсетевые протоколы позволяют организовать пересылку сообщений между разными глобальными сетями. Самый популярный из них IP (Internet Protocol) задействован в гигантской мировой суперсети Internet.

Важную роль в глобальных сетях играют, так называемые, маршрутизаторы (роутеры) - мощные компьютеры или специализированные интеллектуальные устройства, соединяющие между собой различные сети или участки сети. Впрочем, иногда маршрутизатором называют и программу, функционирующую на сетевом компьютере.

3.3. ЭЛЕКТРОННАЯ ПОЧТА

Исторически первый и наиболее распространенный вид работы в телекоммуникационных сетях - **межперсональный обмен текстовыми сообщениями**, известный под названием «электронная почта» (или E-mail). Как и при обычной почтовой связи, здесь происходит обмен сообщениями, но не на бумаге, а в виде файлов. Преимущества электронной почты над обычной велики: многократно большая скорость доставки информации (так, сообщение из России в США обычно доставляется не более, чем за 2 часа), компьютерная подготовка сообщений, передача информации в виде, допускающем последующую ее компьютерную обработку получателем (редактирование, помещение в различные документы, базы данных и т.д.).

Система электронной почты организуется как совокупность региональных узловых станций, периодически связывающихся друг с другом для обмена корреспонденцией, для чего могут использоваться различные каналы связи. Для того, чтобы попасть с машины А на машину В, сообщение может проходить через несколько промежуточных узлов. На каждом узле работают специальные программы, которые получают сообщение и разбираются, куда его отправлять дальше.

Абоненты электронной почты обслуживаются «электронными узлами связи». Для обмена корреспонденцией между абонентом и узлом, как правило, используется обычная телефонная линия.

Абонентская станция состоит из персонального компьютера и модема. Каждый пользователь имеет свой почтовый ящик с уникальным адресом. Все письма, посланные по этому адресу, попадают в почтовый ящик пользователя. Пользователь может просмотреть, уничтожить или сохранить письма. Естественно, любой пользователь может послать письмо по любому адресу или сразу по нескольким адресам.

В отличие от других способов обмена информацией между ЭВМ, система электронной почты дает возможность такого обмена с любой удаленной машиной при фактической связи только с ближайшим узлом системы. Послание хранится на сервере получателя до того момента, когда он выйдет на связь. При этом, может возникнуть значительная транспортная задержка: от нескольких часов до нескольких суток.

Сообщение состоит из текста, который надо передать адресату, и заголовка, который приписывается в начале сообщения и содержит несколько строчек необходимой информации об этом сообщении: дату отправления, адрес, обратный адрес, тему сообщения и другие. Рассмотрим пример почтового сообщения:

```
From insvch!kiaae!gateway.mgn.chel.su!mgpi.mgn.chel.su!mgpi Wed Jan 11
11:26:32 1995
Received: by vgpi.voronezh.su (UUPC/a v5.09gamma, 14Mar93); Wed, 11 Jan
1995 11:26:32 +0300
Received: by insvch.voronezh.su; Tue, 10 Jan 95 17:46:15 +0300
Received: from mgn.chel.su (gateway.mgn.chel.su) by sequent.kiaae.su with
SMTP id AA05915
(5.65.kiaae-2 for
<mogavgpi.voronezh.su>); Tue, 10 Jan 1995 17:20:08 +0300
Received: from mgpi.UUCP by mgn.chel.su with UUCP id TAA02156; (8.6.5)
Tue, 10 Jan 1995
19:17:47+0500
Received: by mgpi.mgn.chel.su (UUPC/@ v5.09gamma, 14Mar93); Tue, 10 Jan
1995 15:26:43 +0500
To: mog@vgpi.voronezh.su Message-Id: <AAY3c41qF3@mgpi.mgn.chel.su>
Organization: Pedagogical Institute
From: mgpi@mgpi.mgn.chel.su (Vadim Bavazitov) Date: Tue, 10 Jan 95
15:26:42 +0500
X-Mailer: BML [MS/DOS Beauty Mail v.L36]
Return-Receipt-To: mgpi@mgpi.mgn.chel.su
Subject: Re: Поздравления с Новым Годом!
Lines: 5
```

Уважаемый Александр Владимирович!

Поздравляю Вас с наступившим Новым Годом! Я надеюсь, Вы не забыли о моей просьбе (об АСУ ВУЗ) .

С уважением Баянзов В.З.

Здесь первые четырнадцать строчек составляют заголовок. Заметим, что каждая из строк заголовка имеет вид

название:текст

Названия строк заголовка расшифровываются так:

Received:

- отметка о прохождении через машину (почтовый штемпель). У нашего письма таких отметок пять, значит, по пути оно прошло через пять машин, и каждая из них обозначила, когда оно проходило.

Date:

- время отправления письма; оно указывается в стандартном формате, поскольку большинство почтовых систем умеют сортировать сообщения по времени, если потребуется.

From:

-адрес отправителя и обратный адрес, отделен угловыми скобками. Message-Id:

- внутренний идентификатор сообщения: присваивается почтовой системой отправителя. Каждому письму присваивается уникальный - единственный в мире! - идентификатор. Его можно использовать для ссылок на письмо, как исходящий номер.

To:

- адрес получателя.

Subject:

- тема сообщения. Пометка Re: обозначает, что это сообщение - ответ (от слова reply) на другое сообщение. У исходного сообщения и у ответа строка Subject: одна и та же. При составлении автором ответа почтовая система автоматически взяла тему из исходного сообщения. Это удобно, когда идет длинный разговор на одну тему. Можно потребовать, чтобы почтовая система отсортировала сообщения по темам, и освежить в памяти предыдущие фразы этого разговора. Составляя сообщение, надо стараться в этой строке указать короткое и как можно более информативное описание темы сообщения.

Status:

- статус сообщения; почтовая система помечает для себя, что сообщение уже прочитано, чтобы второй раз не предложить его как новое.

Бывает еще несколько видов строк заголовка (не все они обязательно должны быть). Некоторые строки почтовые системы добавляют автоматически (Received:, Date:), другие задает сам автор письма (To:, Subject:).

Остановимся более подробно на том, как указать в сообщении адрес, чтобы почтовые системы его поняли и доставили письмо по назначению. Рассмотрим адрес из приведенного выше примера:

mog@vgpi.voronezh.su

Он состоит из адреса машины и имени адресата, которое отделено знаком @. Слева от @ стоит имя адресата, точнее, имя файла - почтового ящика на его машине, из которого он забирает

письма. Обычно пользователь называет свой почтовый ящик тем же именем, под которым он зарегистрирован в системе. Чаще всего это имя, фамилия или инициалы.

Часть справа от @ называется доменом и описывает местонахождение этого почтового ящика (машину или организацию). Рассматривая домен справа налево и разбив его по точкам на отдельные слова, получим поддомены. поочередно уточняющие. где этот почтовый ящик искать. Самый правый поддомен (в нашем случае su) называется доменом верхнего уровня и чаще всего обозначает код страны, в которой находится адресат. Код su сохранился от названия «Советский Союз», а, например, код uk обозначает Великобританию, fr - Францию и т.д.

Домен верхнего уровня - не всегда код страны. В Соединенных Штатах встречаются такие, например, домены верхнего уровня, как edu - научные и учебные организации, или gov - правительственные учреждения: lamaster@george.arc.nasa.gov. Также можно встретить домен верхнего уровня, обозначающий название сети, в которой находится адресат, например, **bitnet**: DLV@cunyvmssl.bitnet.

Поддомены, расположенные правее домена верхнего уровня, уточняют положение адресата внутри этого домена. В нашем первом примере mog@vgpi.voronezh.su поддомен voronezh обозначает город Воронеж, а vgpi - организацию в Воронеже (Воронежский государственный педагогический институт).

Сеть RELCOM поддерживает стандарт RFC822, поэтому адрес пользователя на машине в сети RELCOM можно писать в доменной форме. У всех адресов в RELCOM домен верхнего уровня - это код страны su (или ru от Russia - Россия). Домены первого уровня обычно обозначают город, например, msk (Москва), spb (Санкт-Петербург), nsk (Новосибирск); встречаются домены первого уровня, обозначающие организацию (kiaa, demos).

С помощью электронной почты можно пересылать не только текст, но и двоичные файлы - программы, графические файлы, архивы и т.д. Поскольку, гарантируется прохождение по сетям лишь символов с кодами от 0 до 127 (и лишь в некоторых случаях до 255), двоичные файлы должны быть перекодированы в текст из символов в этом диапазоне кодов. Распространенная система кодировки предоставляется утилитами uencode.exe и udecode.exe.

Запуск программы uencode осуществляется командой с параметрами:

uencode [-<ключ>] <имя двоичного файла> [<имя файла-результата>] После запуска создается текстовый файл примерно с таким содержанием:

```
section I of uencode 5.25 of file uencode.exe by R.E.M.
begin 644 my_prog.exe
M35J='1, '0' ' ' (4) __]:-@#_4@' ' ' '\!4$M,251%($-0<' (N(>$Y
M.3'M.3 (@4$M705) % ($EN8RX@06QL(%)I9VAT<R!297-E<G9E9<'!0!1T'
...

L7!QI'&X<QR&' (L<EiR9'+<PlP&' '#G' ?H='A]A) #TE5B\'TP<'0' "DIP'

end
sum -r/size 37949/13297 section (from «begin» to «end»)
sum -r/size 38958/9629 entire input file ;
```

Здесь в первой строке содержится информация о версии перекодирующей программы и номере секции перекодированного файла (большие файлы разбиваются на секции для удобства пересылки по электронной почте).

В строке со словом **begin** указывается имя исходного двоичного файла. Затем, до слова **end**, идут символы-коды. В конце письма приводятся контрольные суммы для проверки правильности передачи.

Если не задано имя файла-результата, он получает имя исходного файла с номером секции в конце и расширение uce. Необязательные ключи позволяют управлять размером перекодированных секций, форматами перекодировки и т.д. Подсказка к программе появляется при ее запуске без параметров.

Обратное преобразование - в двоичный формат - выполняется с помощью программы **undecode.exe**.

Вызов этих утилит происходит автоматически при работе со специализированными программами для подготовки и чтения электронной почты.

В настоящее время кодировщик uencode вытесняется программой **base64**, реализующей новый протокол перекодировки MIME, обладающий рядом новых возможностей.

Помимо персональных сообщений по электронной почте от одного человека другому, важную роль играют **телеконференции**, реализующие принцип общения «многие-со-многими». Идея телеконференции заключается в том, что все, кто имеет что-то сказать, посылают сообщения, как бы вывешивают записки на доску объявлений, и каждый может читать те записки, которые его интересуют, пропуская неинтересные. Поскольку ежедневно рассылается огромное количество сообщений, один человек не может не только их все читать, но даже просматривать все названия. Поэтому каждый участник конференции заранее отмечает, какие темы его интересуют, и ему приходят сообщения только на эти темы.

Пользователям сети RELCOM в каждый момент доступны сотни постоянно или временно действующих телеконференций. Некоторые из них доступны пользователям всех узлов сети (например, конференция **relcom.education** по проблемам образования), некоторые - лишь в пределах региона (например, в Пермской области конференция **perm.education**). Есть телеконференции коммерческого и некоммерческого характера, различающиеся оплатой за получаемую информацию. Часто однородные по тематике телеконференции объединяют в группы, а внутри них выделяют подгруппы. Например, большая группа «вычислительная техника» содержит подгруппы «языки программирования», «операционные системы», «компьютерные вирусы» и т.д.

Имена news-групп состоят из нескольких слов, разделенных точками. Первое слово обозначает широкую область, к которой относится news-группа, а остальные слова последовательно уточняют ее тему. Например, в имени news-группы **comp.lang.pascal** слово **comp** обозначает, что тема этой группы связана с компьютерами, слово **lang** - с языками программирования, а слово **pascal** определяет язык программирования Pascal.

По телеконференции можно получать либо полные тексты сообщений (статьи), которые в нее посланы, либо лишь, так называемые, артиклы (темы статей) - краткие их содержания, по которым впоследствии можно заказывать сами статьи.

Для того, чтобы получать сообщения той или иной группы, читатель должен на эту группу подписаться - послать на узел сети (или сервер этой конференции) соответствующее сообщение. Например, в RELCOM'e для этого надо отправить по адресу сервера новостей либо команду

subscribe <имя_конференции>

- заказ тем статей указанной конференции, либо команду feed <имя_конференции>
- заказ полных текстов статей.

Сообщения внутри каждой news-группы пронумерованы в порядке поступления, по этим номерам к ним можно обращаться, вызывать полные тексты. Для этого в адрес сервера телеконференции надо послать письмо, содержащее команды

group <имя_конференции>
art <номер_статьи>

Необходимо иметь в виду, что время хранения статей на сервере конференции ограничено сроком от одной недели до месяца. После истечения этого срока статья удаляется.

Для отправки своего сообщения в конференцию надо сформировать файл с текстом сообщения (в режиме подготовки сообщений в телеконференцию), занести его заголовок в поле **Subject**, по которому получатель обычно судит, стоит ли заказывать сообщение целиком (если он получает только артиклы), и отправить сообщение на сервер новостей, указав при этом наименование телеконференции или группы телеконференций, в которую желательно поместить сообщение, географическую область желательного распространения статьи. Обычно на узле или в телеконференции есть администратор, который судит о том, следует ли действительно пропустить ваше сообщение, не нарушаются ли в нем юридические и этические нормы, которым обязаны следовать участники телеконференций.

Обсудим **программную поддержку электронной почты**. Возможности электронной почты с точки зрения конечного пользователя поясним на примере двух программ - почтовых оболочек, - одна из которых работает под управлением MS DOS, а другая - Windows. Возможности пер-

вой, **MAIL-2**, отражены в меню, схематически изображенном на рис. 5.4.

Документ	Прием	Отсылка	Почта	Опции	Настройка
Редактировать Просмотреть Копировать Переименовать Удалить Найти Папка Внешняя команда Запустить ОС Распечатать	Письма Новости Сортировка Уведомление о приеме	Письмо Новость Запрос статей Факс Адресная книга Неотправленные Копия почты Файл подписи	Соединение Запуск таймера Статистика	UUENCODE UUDECODE Архиватор таблиц Сетевые новости Электронная подпись Правила пользования Меню пользователя	Соединения Пользователя Новостей Принтера Перезагрузка

Рис. 5.4. Схема меню программы-оболочки электронной почты MAIL-2

Система предлагает главное меню - верхняя строка - и вертикальные «падающие» меню, развивающие каждый из пунктов главного (на рисунке все вертикальные меню изображены одновременно, хотя реально на экране они присутствуют поочередно).

В разделе «Документ» пользователь может создавать письма и новости, используя встроенный редактор (хотя с таким же успехом можно пользоваться любым внешним текстовым редактором), осуществлять стандартную работу с файлами (копировать, удалять, искать во всех имеющихся директориях). Можно создавать тематические папки для архивов. Возможен временный выход в DOS и последующий возврат в данную программу.

В разделе «Прием» можно просматривать полученную корреспонденцию - письма, новости, - копировать их как во внутренний архив, так и в любой внешний каталог, стирать. Можно задать режим автоматической отсылки уведомлений о получении писем или отменить его.

В разделе «Отсылка» производится отправление указанных видов сообщений. Адреса для писем можно либо вводить в интерактивном режиме, либо брать из «Адресной книги»; последнюю можно пополнять и редактировать. Можно просмотреть неотправленные письма и при необходимости их уничтожить.

Для упрощения ввода адресов конкретных абонентов в командах отправки писем в программе предусмотрен электронный справочник адресов электронной почты. Для вызова на экран окна справочника используется команда меню {Адресная книга}. В окне отображается таблица известных пользователю адресов сети. В каждой строке показывается собственно адрес и произвольный комментарий к нему (это обычно фамилия пользователя, телефон и т.п.). В режиме справочника адресов, кроме стандартных команд перемещения, можно внести новый адрес в таблицу. На экране появляется окно диалога с полями «Адрес» и «Комментарий», в которые следует ввести требуемые значения: фамилию, имя, отчество абонента, электронный адрес, место работы, телефон и т.д. Пользователь должен сам заботиться о расширении своего справочника, вводя в него адреса интересующих его абонентов.

Подготовка письма в телеконференцию производится как обычно, но для его отправки вместо команды {Отсылка} {Письма} используется команда {Отсылка} {Новость}. В появившемся окне диалога программа предложит вам ввести имя отправляемого файла, раздел телеконференции и строку-комментарий к письму.

В разделе «Почта» происходит, в частности, соединение с почтовой машиной (узлом). В зависимости от настройки компьютер дозванивается по заложенным в программу номерам телефонов, делая заданное число попыток.

В разделе «Опции» первый пункт - запуск программы кодирования. Это бывает необходимо для передачи по сетям бинарных файлов, а также некоторых текстовых (например, русскоязычных текстов по иностранным сетям). Исходный документ преобразуется в файл с 7-битовой

кодировкой и приобретает специальное расширение .luc, с которым его и следует отправлять. Второй пункт этого подменю - обратное преобразование. Вся деятельность происходит в интерактивном режиме. Следует заметить, что такое стандартное кодирование не рассматривается как самостоятельная шифровка информации, запрещенная российским законодательством. Пункт «Архиватор таблиц» позволяет помещать во внутренний архив программы MAIL-2 письма и документы, пункт «Сетевые новости» позволяет соединиться с сервером новостей по специальной программе обслуживания локальной сети. «Электронная подпись» позволяет сформировать стандартную подпись для Ваших писем, автоматически подключаемую к каждому из них. «Подпись» содержит номера телефонов, факсов, сведения о должности и т.д.

Раздел «настройка» позволяет конфигурировать программу соединения с сервером, подписываться и отказываться от подписки на группы новостей и т.д. В разделе «настройка соединения» пользователь выбирает одну из нескольких возможных программ связи по модему.

Большое распространение в России получил пакет для работы с электронной почтой в среде DOS под названием UUPC.

Среди файлов и директорий системы UUPC, формирующихся при установке этой системы, наиболее важными являются:

\UUPC	-	каталог, в котором размещены составные части пакета UUPC;
\UUPC\READ_ME	-	инструкция по почтовой системе;
WUPCVUUCICO.EXE	-	программа передачи/приема почты, обычно запускаемая с помощью bat-файла uupc.bat ;
\UUPC\BML.EXE	-	программа, позволяющая работать с поступившей почтой и подготавливать новые сообщения;
\UUPC\CONF\	-	каталог, содержащий общие файлы настройки;
\UUPC\CONF\DIALERS	-	описания модемов, известных системе;
\UUPC\CONF\SYSTEMS	-	описание вызываемых узлов сети (телефоны, пароли и т.п.);
\UUPC\CONF\UUPC.RC	-	настройка общих параметров системы UUPC (полное сетевое имя, краткое сетевое имя и т.д.);
\UUPC\CONF\PASSWD	-	список всех пользователей UUPC;
\UUPC\CONF\FORWARD	-	файл с именем администратора почты на данной машине;
\UUPC\MAIL\BOXES\имя_пользователя	-	файл системного почтового ящика.

Все файлы настройки содержат подробный комментарий и могут быть отредактированы любым текстовым редактором для внесения изменений в настройку системы. Кроме того, для каждого пользователя системы создается домашний каталог (например, C:\MOG - для пользователя MOG), который содержит следующие файлы:

PERSONAL.RC	-	параметры системы UUPC для данного пользователя (имя, расположение системных каталогов);
PERSONAL.SIG	-	файл, который автоматически добавляется в конец каждого посланного сообщения, как бы подпись под электронным письмом (если задан такой режим);
MBOX	-	файл для записи уже прочитанных писем (по умолчанию);
FORWARD	-	кому перенаправить почту (файл состоит из одной строки, где указаны одно или несколько сетевых имен, через пробел, а весь список занимает не более 4 кбайт).

В указанный при установке системный каталог записываются команды для облегчения вызова системы:

UUPC.BAT	-	вызов программы связи;
BMAIL.BAT	-	экранная версия программы просмотра и отправки почты.

Передача всех писем (и прием писем, предназначенных для ЭВМ-адресата) осуществляется при запуске программы связи с помощью команды UUPC. Система UUPC будет пытаться связаться с главной машиной и передать и принять накопленную информацию. Если связь не удалась, произойдет повтор набора телефонного номера. Когда все подготовленные сообщения переданы и новая почта принята, программа связи прекращает работу.

В среде **Windows** одна из широко распространенных программ электронной почты - **Outlook Express**. Ее заставка, содержащая главное (графическое) меню, приведена на рис. 5.5.

Собственно, **Outlook Express** представляет нечто большее, чем просто программу просмотра и подготовки электронной почты. Это - среда, поддерживающая различные виды деятельности, например, планирование и организацию личной деятельности и групповых мероприятий, учет. Для этого введены папки «Дневник», «Заметки», «Календарь», «Контакты».

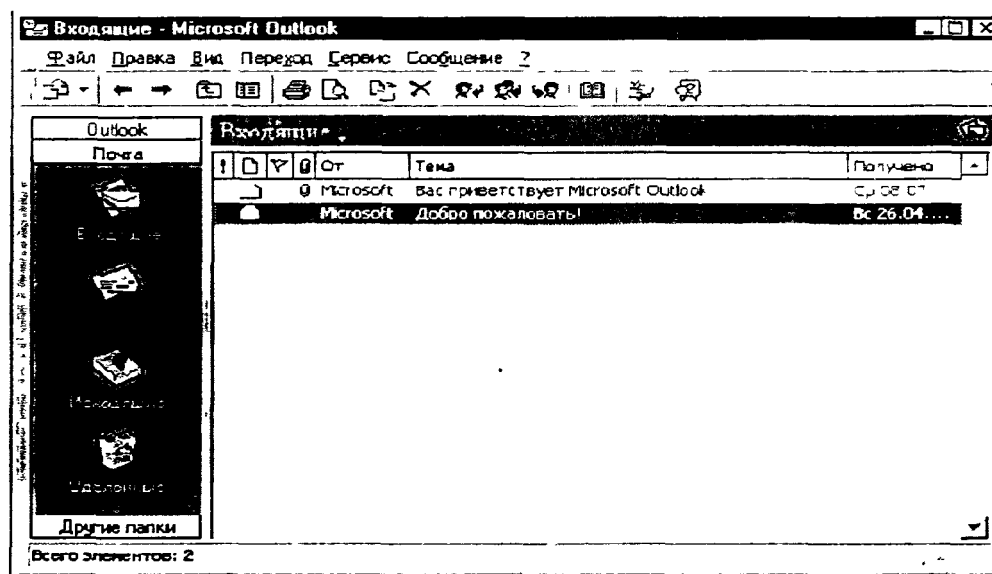


Рис. 5.5. Начальный кадр почтовой программы Outlook Express

Заметки - это электронный вариант бумажного блокнота; здесь можно записывать вопросы, мысли, напоминания и т.п.

Папку «Календарь» удобно использовать для планирования кратковременных мероприятий личного характера, папку «События» для планирования мероприятий, продолжающихся 24 часа и более, например, конференций или отпусков. Планировщик собраний очень удобен, чтобы пригласить людей на собрание и зарезервировать место и оборудование.

Папка «Контакты» предназначена для управления данными о деловых и других контактах. Эти данные можно сортировать и хранить в любом виде; можно хранить по несколько адресов, номеров телефонов или адресов электронной почты для каждого контакта и непосредственно обращаться к их Web-страницам.

И все же главное в **Outlook Express** - это программа электронной почты и чтения новостей, основанная на стандартах Internet.

Для чтения электронной почты с помощью **Outlook Express** необходим доступ к системе почты, использующей протоколы SMTP и POP3 или IMAP, который можно получить у поставщиков (провайдеров) услуг **Internet** в любом регионе. При этом соединение с почтовой системой может происходить как с помощью модема по коммутируемой телефонной линии, так и по локальной сети.

Программу **Microsoft Outlook Express News** можно использовать для чтения новостей в группах электронных досок объявлений, таких, как Usenet; для этого необходимо подключаться к серверу новостей, работающему по протоколу NNTP.

Работа с почтой начинается (и заканчивается) доставкой почты, которая происходит после нажатия на кнопку «Доставить почту». Если имеется постоянная связь с почтовым сервером, доставка происходит сразу, иначе **Microsoft Outlook** сам устанавливает связь с сервером в соответствии с настройками, задаваемыми при установке. Пришедшая почта по умолчанию попадает в папку «Входящие».

Используя список сообщений и область просмотра, можно просматривать перечень сообщений и одновременно читать отдельные сообщения. В любое сообщение можно «войти», щелкнув по его заголовку в списке сообщений, после чего открывается окно выбранного сообщения с соответствующими атрибутами.

В список папок включаются папки почты, серверы новостей и группы новостей; между ними возможно быстрое переключение. Можно добавить свои папки для более удобной организации

сообщений, а затем установить правила папки «Входящие», чтобы поступающая почта, удовлетворяющая заданным условиям, автоматически попадала в конкретную папку.

Для подготовки нового сообщения надо нажать кнопку «Создать сообщение», после чего открывается окно бланка сообщения, имеющего строки ввода адреса для сообщения и его копий, а также темы сообщения.

Интересно, что вводимые адреса автоматически попадают в адресную книгу, откуда их затем удобно выбирать при повторном использовании. Если новое сообщение должно быть ответом на поступившее письмо, следует использовать кнопку «Ответить автору»; в этом случае в бланке сообщения строки адреса и темы заполняются автоматически, а также (если задано в настройках) в текст ответа включается письмо-оригинал, помеченное в каждой строке символом >. Кнопка-команда «Переслать сообщение» позволяет переадресовать полученное письмо кому-либо.

Полезной кнопкой является «Язык». Она позволяет выбрать кодировку просматриваемого сообщения, если сообщение на русском языке пришло в иной (не **Windows**) кодировке (KOI-8, ISO) и имеет поэтому нечитаемый вид.

Меню **Microsoft Outlook** является в значительной степени стандартным для программных продуктов этой фирмы. Оно дублирует команды, вынесенные на кнопки, а также позволяет выполнить поиск, копирование и перемещение сообщений по папкам, а также разнообразные настройки.

Отметим, что меню окна создания сообщения (ответа) отличается от основного меню **Microsoft Outlook**. Так, опция «Вставка» позволяет вставить в сообщение заранее подготовленный текст (в кодировке **Windows**) из файла с расширением **.txt** или «вложить» в сообщение двоичный файл с перекодировкой в текстовый формат по протоколу MIME. При поступлении сообщения с таким вложением пользователю **Microsoft Outlook** оно автоматически разворачивается в двоичный файл, записывается на диск или может быть открыто сразу же с помощью имеющегося в ОС приложения.

Удобные возможности **Microsoft Outlook** предоставляет в работе с группами новостей **Usenet**. Можно провести их поиск по ключевым словам или просмотреть список всех групп новостей, доступных через поставщика **Usenet**. Найдя группу новостей, которую следует регулярно просматривать, надо включить ее в список «Выписанные», чтобы ее всегда можно было быстро найти.

В группе новостей можно, не читая всего списка сообщений, посмотреть отдельное сообщение и все ответы на него (так называемое «обсуждение»). При просмотре списка для облегчения поиска интересующей темы можно развертывать и свертывать обсуждения. На сообщении можно поставить отметку о том, что оно уже прочитано; можно отсеять сообщения, которые не нужны, или задать отображение только непрочитанных сообщений.

Чтобы эффективнее использовать время подключения к **Internet**, можно загрузить с сервера сообщения или целые группы новостей, а затем спокойно их прочитать, отключившись от поставщика услуг. Можно получить для автономного просмотра одни заголовки сообщений и пометить те из них, которые вы хотите прочитать при следующем подключении. Можно также готовить сообщения в автономном режиме, и отправлять их при очередном подключении.

Специфической разновидностью электронной почты можно считать общение с почтовыми роботами - файловыми серверами (архивами) и другими банками информации. В этом случае электронные письма состоят из специальных команд. Набор команд может несколько отличаться для различных серверов, но, в основном, стандартным является следующий набор:

HELP	- команда обращения за помощью, в ответ на нее сервер высылает письмо, содержащее описание команд сервера;
INDEX	- запрос о структуре информации, хранящейся на сервере, с краткими описаниями файлов;
DIR и LS	- команды, возвращающие содержимое каталогов файлового сервера;
CD <имя_директория>	- команда перемещения по дереву директорий;
GET <имя_файла>	- вызов файла;
QUIT	- окончание командного пакета.

Файловый сервер обычно содержит хорошо упакованные архивы, т.е. двоичные файлы, требующие для передачи по электронной почте конвертирования в текстовый вид, разрезания на части небольшой длины. Также при необходимости можно изменить способ архивирования получаемых файлов. Для этого имеются специальные команды.

Существует множество компьютерных баз и банков данных по различным проблемам человеческой деятельности. Их списки и электронные адреса регулярно публикуются и рассылаются по сетям. Обращаться к серверам файловых архивов надо очень осторожно. Дело в том, что многие файловые серверы являются коммерческими и стоимость получения информации с них составляет порядка 0.03\$ за килобайт. Заказав по электронной почте архив какой-нибудь большой системы можно не только перегрузить сеть большим количеством информации, но и получить в результате счет на оплату значительной денежной суммы.

3.4. ON-LINE INTERNET

3.4.1. Адресация и виды информации в Internet

По данным на 1995 г. через **Internet** связано более 5 млн. компьютеров, и число их очень быстро растет. Internet поддерживает все существующие виды работ, возможных в телекоммуникационных сетях. Эта суперсеть, охватившая весь мир, представляет из себя совокупность многих (более 2000) сетей, поддерживающих единый протокол TCP/IP (**Transmission Control Protocol/Internet Protocol**).

Основой **Internet** является система так называемых IP-адресов. Каждый компьютер, включенный в **Internet** (а точнее каждый сетевой интерфейс) получает уникальный в рамках всего Internet адрес (адресами ведают национальные комитеты *Internet*). IP-адрес - это 4-байтовая последовательность, каждый байт записывается в виде десятичного числа. Например, 195.19.19.19 - адрес одной из машин Воронежского государственного педагогического университета.

IP-адрес состоит из двух частей: адреса сети и номера хоста. Под хостом следует понимать не только компьютер в сети, но и вообще любое устройство, которое имеет свой сетевой интерфейс.

Существует несколько классов IP-адресов. Эти классы отличаются друг от друга количеством битов, отведенных на адрес сети и адрес хоста. При разработке структуры IP-адресов предполагалось, что они будут использоваться по разному назначению. Адреса класса А предназначены для использования в больших сетях общего пользования. Адреса класса В предназначены для использования в сетях среднего размера (сети больших предприятий и учреждений, вузов). Адреса класса С предназначены для использования в сетях с небольшим числом компьютеров (табл. 5.6).

Таблица 5.6
Классы IP-адресов

Класс	Диапазон значений первого байта	Возможное число сетей	Возможное число узлов
A	1-126	126	16777214
B	128- 191	16382	65534
C	192-223	2097150	254

Протокол IP описывает три основные компоненты IP-технологии: формат IP-пакета, формат IP-адреса и способ маршрутизации IP-пакетов. Структура IP-пакета отражена на рис. 5.6.

В настоящее время используется четвертая версия протокола, следовательно, поле *версия* принимает значение «4». Сам заголовок, включая необязательные опции и символы заполнения, имеет длину, равную значению поля *длина IP-заголовка*. Длина исчисляется в 32-битовых словах. Поле *тип сервиса* определяет способ обслуживания пакета в конкретных сетях и, главным образом, связано с возможностью задержки (**delay**) пакета в сети. Поле *общая длина* определяет длину IP-пакета без заголовка. Длина измеряется в октетах (байтах). Поле *идентификация* предназначено для помощи при «сборке» сообщения. Поле *флаги* определяет место датаграммы в сообщении

(первая, последняя и т.п.). Поле *смещение фрагмента* определяет смещение датаграммы относительно начала сообщения. Поле *время жизни* предназначено для определения срока, после которого пакет должен быть удален из сети. Поле *протокол* определяет тип датаграммы. Поле *контрольная сумма* служит для идентификации повреждений пакета при передаче. Контрольная сумма вычисляется только по заголовку пакета. Поле *адрес получателя* - IP-адрес места назначения. Поле *адрес отправителя* - IP-адрес отправителя. *Опции* могут иметь переменную длину и обычно применяются для трассировки пакетов, обеспечения безопасности. *Заполнитель* применяется для выравнивания заголовка на 32-битовую границу.

Версия (4 бита)	Длина IP заголовка	Тип сервиса (8 битов)	Общая длина (16 битов)	
Идентификация (16 битов)			Флаги (3 бита)	Смещение фрагмента
Время жизни (8 битов)	Протокол (8 битов)		Контрольная сумма (16 битов)	
Адрес отправителя (32 бита)				
Адрес получателя (32 бита)				
Опции			Заполнитель	
Данные				

Рис. 5.6. Структура IP-пакета

Поскольку числовая адресация неудобна для использования ее человеком, в сети **Internet** числовым адресам ставятся в соответствие буквенные доменные. Поддержка доменных адресов производится с помощью DNS (Domain Name System) - серверной программы, которая обеспечивает поиск доменных имен и IP-адресов узлов сети в зоне ответственности сервера, устанавливает между ними соответствие.

Для проверки прохождения IP-пакетов и доступности тех или иных машин используется программа **Ping**. В качестве параметра команды запуска этой программы указывается IP-адрес или доменный адрес нужного сервера. В ответ Ping сообщает число отосланных пакетов, число потерянных пакетов и время прохождения пакета. По этим данным можно судить о качестве связи в сети.

На базе протоколов TCP/IP реализованы другие прикладные протоколы **Internet**, составляющие основу сервиса в сети.

Для поддержки электронной почты в Internet разработан специальный протокол SMTP (Simple Mail Transfer Protocol), устанавливающий связь с машиной-получателем и передающий почту в режиме **on-line** (в отличие от описанных выше протоколов, работающих в режиме **off-line**).

В Internet используется стандарт MIME, описывающий тело почтового сообщения. В отличие от стандарта RFC-822 для электронной почты, обеспечивающего передачу только текстовой информации в 7-битной кодировке ASCII (двоичные файлы при этом при передаче приходилось конвертировать в текстовую форму программой UUDECODE) стандарт MIME позволяет передающей машине описывать в заголовке электронного письма тип информационных единиц, составляющих письмо, для обеспечения их правильной интерпретации машиной-получателем. В настоящее время используются семь типов данных, которые можно передавать в теле письма:

- текст (text);
- смешанный тип (multipart);
- почтовое сообщение (message);
- графический образ (image);
- аудио-информация (audio);
- фильм или видео (video);
- приложение (application).

Для каждого из этих типов имеется по несколько подтипов данных. Однако, несмотря на такое богатство типизации, стандарт MIME предусматривает использование новых типов данных

в почтовых сообщениях.

Кодирование информации происходит чаще всего с помощью, так называемого, 64base - набора символов.

Помимо межперсональной электронной почты и телеконференций, Internet обеспечивает доступ к огромным информационным ресурсам, среди которых важнейшими являются

- система файловых архивов FTP и поисковых серверов ARCHIE;
- банки информации WWW;
- базы данных Gopher;
- базы данных WAIS;
- справочные службы WHOIS.

3.4.2. Доступ к информации в Internet

Главный режим доступа к информационным ресурсам Internet **-on-line**, однако получать информацию из этих ресурсов можно и в режиме **off-line** с помощью простейшей электронной почты, поскольку на всех видах серверов в Internet обычно имеются почтовые роботы, обрабатывающие почтовые сообщения, содержащие команды сервера.

Система **файловых архивов FTP** - это огромное распределенное хранилище всевозможной информации, накопленной за последние 10-15 лет в Сети. Любой пользователь может воспользоваться услугами анонимного доступа к этому хранилищу и скопировать интересующие его материалы. Объем программного обеспечения в архивах FTP составляет терабайты информации (приставка «тера» соответствует множителю 1012), и ни один пользователь или администратор сети просто физически не может обозреть эту информацию. Кроме программ в FTP-архивах можно найти стандарты **Inlernet-RFC**, пресс-релизы, книги по различным отраслям знаний (и особенно по компьютерной проблематике) и многое другое.

Практически любой архив строится как иерархия директорий. Многие архивы дублируют информацию из других архивов (так называемые «зеркала» - mirrors). Для того, чтобы получить нужную информацию, вовсе не обязательно ждать, когда информация будет передана из Австралии или Южной Африки, можно поискать «зеркало» где-нибудь ближе, например в Финляндии или Швеции. Для этой цели существует специальная программа Archie, которая позволяет просканировать FTP-архивы и найти тот, который устраивает пользователя по составу программного обеспечения и коммуникационным условиям.

Кроме того, FTP - это интерфейс пользователя при обмене файлами по одноименному протоколу. Программа устанавливает канал управления с удаленным сервером и ожидает команд пользователя. Идентификатор удаленного сервера указывается в команде интерфейса **open**. Если команда ftp работает с пользователем и ожидает его команд, то на экране отображается приглашение

```
ftp>
```

Первой командой сеанса является команда **open**. По этой команде открывается сеанс работы с удаленным сервером:

```
ftp>open ftp.kiae.su
```

После выдачи такой команды последуют запросы идентификации пользователя. Зарегистрировать пользователя можно и по команде **user**:

```
ftp> user anonymous
```

Следующими по важности командами являются команды **cd** и **ls (dir)**. Назначение этих команд достаточно прозрачно и понятно всем пользователям - навигация по дереву файловой системы и просмотр содержания каталогов. Здесь следует посоветовать пользоваться при просмотре каталогов командой **ls** с дополнительными параметрами:

```
ftp>ls -FC
```

И, наконец, самыми важными являются команды приема/передачи данных **get**, **put**, **mget**, **input** и **bin**. По командам **get** и **put** можно принять или передать один файл:

```
ftp> get README.TXT
```

Команды **mget**, **input** предназначены для приема/передачи набора файлов:

```
ftp> mget *.gz
```

Для **поиска** информации на FTP-серверах создавались поисковые серверы ARCHIE. Ближайшим таким сервером является **archi.funet.fi** в Финляндии. Однако, в настоящее время функции поиска перешли к поисковым WWW-серверам, рассматриваемым ниже.

«**Мировая паутина**» **WWW** (от английских слов World Wide Web - «Мировая паутина») - это распределенная информационная система мультимедиа, основанная на гипертексте. Информация хранится в ней на огромном множестве объединенных в сеть серверов - компьютерах с соответствующим программным обеспечением. Пользователи, или клиенты, имеющие доступ к сети, могут получать эту информацию, используя специальное программное обеспечение - программы просмотра WWW-документов (так называемые, WWW-броузеры). Информация представлена в виде документов, которые содержат ссылки на другие документы, хранящиеся как на том же самом сервере, так и на других, т.е. образует гипертекст. На экране компьютера элементы, имеющие ссылки на другие документы, тем или иным образом (цветом, подчеркиванием) выделены. Пользователь, работающий с программой просмотра, может связаться с соответствующим сервером сети и получить документ, на который имелась ссылка. Для этого ему достаточно переместить указатель мыши на выделенный элемент и нажать клавишу мыши. Программа просмотра при этом посылает запрос серверу, хранящему документ.

Взаимодействие клиент - сервер при работе на WWW происходит по протоколу, который называется **HyperText Transport** (или **Transfer**) **Protocol** - сокращенно HTTP. Для получения WWW-документов с сервера в запросе обязательно указывается протокол, адрес сервера, имя директории на сервере и файла с необходимым документом. Эта комбинация получила название **Uniform Resource Locator** (сокращенно - URL или просто **Location**). Типичный URL выглядит следующим образом:

<http://www.vspu.ac.ru/de/index.htm>

WWW-клиенты могут общаться с серверами, поддерживающими, кроме **http**, протоколы **ftp**, **gopher** и др. Если имя директории или файла не указано в URL, сервер вышлет файл **index.htm** или **welcome.htm**.

В настоящее время получили распространение более десяти программ просмотра WWW-документов. Из них наиболее популярными являются программы **Netscape Navigator** и **Microsoft Internet Explorer**, работающие в среде **Windows**.

Информация на WWW включает в себя не только текст, но и графику, видео и звук. На сервере могут также быть запущены и прикладные программы.

Для **поиска** информации в «паутине» имеется развитая сеть поисковых WWW-серверов, использующих различные методы индексации информации в сети для быстрого контекстного поиска, причем поиск может выполняться для всех видов серверов сети - WWW, FTP, GOPHER и др. Одними из самых популярных остаются следующие поисковые серверы: **www.yahoo.com**, **www.altavista.digital.com**, а для поиска русскоязычных ресурсов - **www.rambler.ru**.

При входе на тот или иной поисковый сервер на экране обычно появляется окно для ввода слова или фразы, вхождения которых надо искать (рис. 5.7). Для поиска нужной информации в это окно следует ввести ключевые слова (конъюнкция условий поиска обычно обозначается знаком &), указать язык поиска, вид серверов и запустить поиск. Возможно, в течение нескольких минут будет найдено несколько сотен или даже тысяч вхождений искомых ключевых слов в документы

во всей сети, которые придется просматривать «вручную» и из которых большинство окажется малоинформативными.

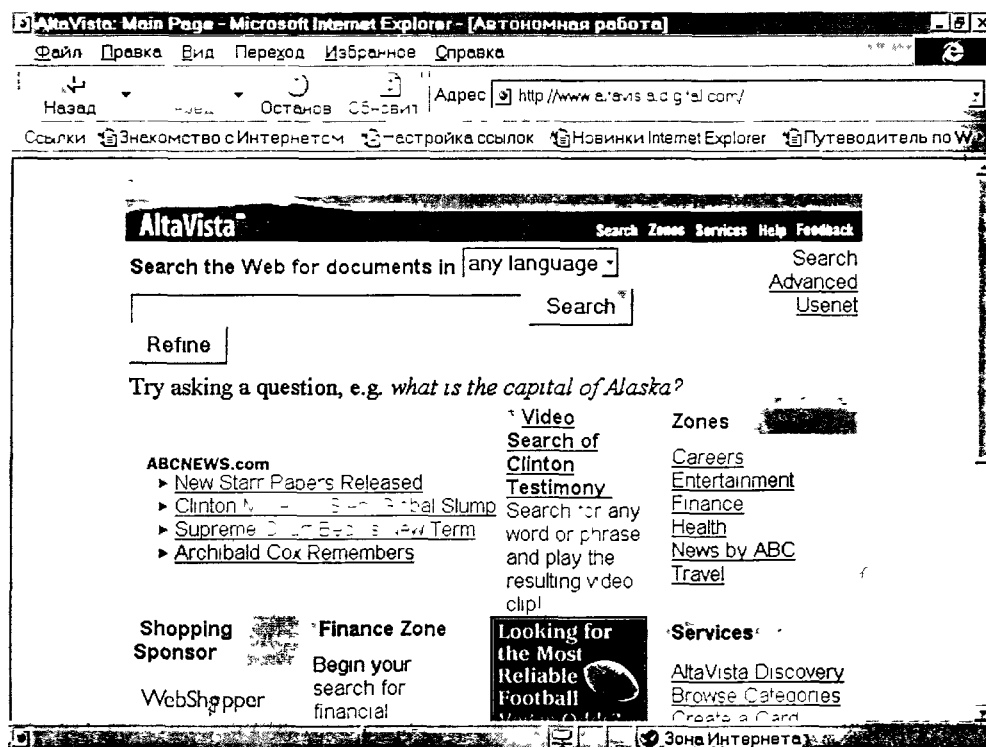


Рис. 5.7. «Домашняя» страница поискового сервера Alta Vista в браузере Internet Explorer

Другой вариант поиска - через «дерево» рубрик. Однако и в этом случае поиск нужного сервера или документа может оказаться трудоемким или вообще безрезультатным.

3.4.3. Язык разметки гипертекстов HTML

Документы, хранящиеся на WWW, подготавливаются на специальном языке HTML (**Hyper-text Markup Language** - «Язык разметки гипертекстов»). В HTML текст кодируется в ASCII и поэтому может быть создан и отредактирован в любом текстовом редакторе. Кроме самого текста в документах имеются флаги разметки, которые интерпретируются клиентской программой просмотра (браузером) и позволяют воспринимать документ именно как гипертекст.

Любой гипертекст похож на книгу и может быть разбит на отдельные структурные элементы:

- собственно текст;
- главы, параграфы, пункты, подпункты;
- абзацы.

Для каждого из этих элементов в HTML существуют определенные стили, описывающие, в каком виде пользователь увидит текст на экране. Пусть мы создали файл **minihtml.html**:

```
<BODY>
<TITLE>Пример HTML-текста</TITLE>
<H1>Глава 1</H1>
<H2>Параграф 1.</H2>
Добро пожаловать в HTML!
Это пример гипертекста.<P>
<H2>Параграф 2.</H2><P>
<A HREF="minihtml.html">пример HTML-текста</A>
</BODY>
```

Из данного примера видно, что

- заголовок документа начинается с <TITLE> и заканчивается </TITLE>;
- заголовок первого уровня (главы) выделяется символами <H1> и </H1>;

•заголовки последующих уровней (параграфы, пункты, подпункты и т.п.) -символами <Hx> и </Hx>, где x - числа 2,3, ...

• абзац - символами <P>.

Не все стили поддерживаются всеми WWW-броузерами. Если программа просмотра не поддерживает стиль, то она его игнорирует.

Основной текст отделяется от сопроводительного символами

<BODY> </BODY>

Каждый HTML-документ должен иметь заголовок, который показывается отдельно и используется, прежде всего, для идентификации документа (например, при поиске). Заголовок должен описывать цель документа и содержать не больше 5-6 слов. Практически во всех броузерах заголовок документа виден в верхней части экрана (окна).

Для выделения заголовка служат символы

<HEAD><TITLE>Заголовок</TITLE></HEAD>

HTML имеет шесть уровней заголовков, имеющих номера с 1 по 6 (заголовок первого уровня является заголовком высшего уровня). По сравнению с нормальным текстом, заголовки выделяются шрифтом - размером и толщиной букв. Первый заголовок в каждом документе должен быть выделен <H1>. Синтаксис заголовков:

<Hу> Текст заголовка</Hу>

где у - число от 1 до 6, определяющее уровень заголовка.

В отличие от документов, в большинстве текстовых процессоров прерывания строк и слов в HTML-файлах несущественны. Обрыв слова или строки может происходить в любом месте в исходном файле, при просмотре это прерывание будет проигнорировано. Напомним, что в нашем примере, первый параграф был представлен как

<H2>Параграф 1.</H2>

Добро пожаловать в HTML!

Это пример гипертекста.<P>

В исходном файле два предложения находятся на двух разных строках. Программа просмотра игнорирует это прерывание строки и начинает новый абзац только после знака <P>.

Главное преимущество HTML состоит в его способности связываться с другими документами. Броузер выделяет (обычно цветом и/или подчеркиванием) ключевые слова, являющиеся гипертекстовыми ссылками (гиперссылками). Описывается ссылка на другой документ следующим образом:

Текст, который будет служить как обращение к другому документу. Приведем пример такой гипертекстовой ссылки:

пример HTML-текста

Здесь ключевые слова «Пример HTML-текста» являются гиперссылкой на файл minihtml.html, который находится в той же директории, что и текущий документ. Также можно ссылаться на документ в любой директории, описав к нему полный путь. Например, ссылку на файл *NJStais.html*, лежащий в поддиректории *AtlanticStates*, можно описать так:

New Jersey

Это - так называемые относительные ссылки. Можно также использовать абсолютное имя

файла (полный путь). В общем случае, использование ссылки по абсолютному имени файла более предпочтительно.

Ссылки на документы, хранящиеся на других серверах, записываются с указанием их URL:

```
<A href="http://www.simtel.ru/news/snews.http">News</A>
```

Гиперссылки могут также использоваться для соединения с определенными разделами документов. Предположим, необходимо соединить документ А с первой главой документа В. Для этого надо создать именованную гиперссылку в документе В:

Здесь вы можете увидеть

```
<A NAME = "Глава 1">Главу 1</A> текст первой главы.
```

Теперь, описывая ссылку в документе А, надо включить не только имя файла *documentB.html*, но также и имя гиперссылки, отделяемое символом (#):

Здесь вы можете увидеть текст

```
<A HREF="documentB.html#Глава"> Главы 1 </A> документа В
```

Теперь «кликнув» в слово «Главы I» в документе А, можно перейти непосредственно в Главу 1 в документе В.

Техника соединения с разделами текущего документа аналогична описанной выше, только опускается имя файла. Вот, например, связь с Главой 1 внутри того же самого файла (Документ В):

```
Это <A HREF = "#Глава1">Глава 1</A> текущего документа.
```

Для более сложных документов HTML имеет некоторые дополнительные возможности форматирования. Так, HTML поддерживает нумерованные и нумерованные списки. Ненумерованный список.

```
<UL><LI>список пунктов </UL>
```

Например:

```
<UL>
<LI> яблоки
<LI> бананы
</UL>
```

что дает на экране:

- * яблоки
- * бананы

Нумерованный список идентичен ненумерованному списку, только вместо используется .

Например:

```
<OL>
<LI> апельсины
<LI> персики
<LI> виноград
</OL>
```

что дает на экране:

1. апельсины
2. персики
3. виноград

Программа просмотра автоматически нумерует элементы такого списка. Списки могут быть произвольно вложены, хотя практически было бы разумнее ограничиться тремя уровнями вложенных списков.

Приведем пример вложенных списков:

```
<UL> <LI> Крупные города России;  
<UL>  
<LI> Москва  
<LI> Санкт-Петербург  
</UL>  
<LI> Крупные города Украины:  
<UL>  
<LI> Киев  
</UL>  
</UL>
```

что дает на экране:

```
* Крупные города России:  
+ Москва + Санкт-Петербург  
* Крупные города Украины:  
+Киев
```

Флаг <ADDRESS> используется для того, чтобы определить автора документа и способы контакта с ним (например, e-mail адрес). Обычно это последний пункт в файле. Например, последняя строка документа может выглядеть так:

```
<ADDRESS> Web-master Воронежского педуниверситета den@vspu.ac.ru</ADDRESS>
```

что дает на экране: Web-master Воронежского педуниверситета den@vspu.ac.ru

Можно описывать специальными стилями отдельные слова или предложения. Имеются два типа стилей: логический и физический. Логические стили определяют текст согласно заданному значению, в то время как физические стили определяют некоторые участки текста.

Логические стили:

<DFN>	служит для описания определений. (Это определение.)
	служит для выделения слов. (Это выделенное слово.)
<CITE>	служит для выделения заголовков книг, фильмов, цитат и т.п. (Это цитата.)
<CODE>	служит для выделения программных кодов, текстов программ и т. п. Изображается шрифтом фиксированной ширины. (The <stdio.h> header file)
<KBD>	используется для ввода с клавиатуры пользователя. Может быть изображено жирным шрифтом (но в большинстве browser изображается специальным шрифтом) (Введите passwd)
<SAMP>	используется для машинных сообщений. Изображается шрифтом фиксированной ширины. (Segmentation fault: Core dumped)
	служит для ОСОБОГО выделения слов. Обычно выделяется жирным шрифтом. (Это очень важно.)
<VAR>	используется для символьных переменных. (Это переменная.)

Существуют физические способы выделения - автор задает стиль написания текста, описывая шрифт в исходном HTML-документе. Можно задать:

, 	жирный шрифт (это жирный шрифт)
<I>, </I>	наклонный шрифт (это <i>наклонный шрифт</i>)
<TT>, </TT>	фиксированный шрифт (шрифт заданной ширины) (это fixed шрифт)

Используя флаг <HR> можно разделить текст горизонтальной чертой:

Этот текст снизу от линии.

Большинство программ просмотра могут показывать рисунки X Bitmap (XBM) или GIF формата вместе с текстом. Чтобы включить рисунок, надо описать гиперссылку на него:

```
<IMG SRC=image_URL>
```

где *image_URL* - URL *.gif* или *.xbm* файла, содержащего рисунок. Таким образом, синтаксис ссылки на рисунок аналогичен синтаксису гиперссылки HREF. Форматирование положения рисунка задается включением в гиперссылку пункта "ALIGN = ":

```
<IMG ALIGN=top SRC=image_URL>
```

Также возможны типы выравнивания:

```
ALIGN = MIDDLE
```

```
ALIGN = CENTER
```

Поскольку пересылка картинки замедляет загрузку основного WWW-документа, рекомендуется помещать рисунок в отдельный документ, написав на него гиперссылку. В этом случае пользователь сам может решить - смотреть или не смотреть ему этот рисунок;

```
<A HREF = ima.ge_URL>Здесь</A> вы можете увидеть рисунок.
```

Рассмотрим более длинный пример HTML-документа:

```
<HEAD>
<TITLE>;Более длинный пример</TITLE>
</HEAD>
<BODY>
<H1>Более длинный пример</H1> Это простой HTML-документ.
Это первый абзац. <P>
Это второй абзац, он демонстрирует некоторые возможности HTML по вы-
делению слов.
Это слово написано <I> наклонным </I> шрифтом.
Это слово написано <B> жирным </B> шрифтом.
Здесь вы можете увидеть картинку:
<IMG SRC="LOGO.GIF"><P>
Это третий абзац, он демонстрирует использование гиперссылок.
Это гиперссылка на файл minihtml.html, содержащий простой пример
HTML-документа:
<A HREF="minihtml.htm">рпНМер HTML-ТеКсра</A>.<P>
<H2>Заголовок второго уровня</H2>
Дальнейший текст будет написан шрифтом фиксированной ширины: <P>
<PRE> On the stiff mig up there Hunches a wet black rook Arranging
and rearranging its feathers in the rain ...
</PRE> Это нумерованный список, состоящий из двух элементов: <P>
<UL>
<LI> смородина
<LI> черника
</UL>
Конец документа. <P>
<ADDRESS> Web-master Воронежского педуниверситета
den@vspu.ac.ru</ADDRESS>
```

3.4.4. Программа-оболочка Internet Explorer

Одним из самых популярных в настоящее время браузеров - программ для просмотра документов WWW-серверов - является **Internet Explorer**, программа, разработанная компанией **Microsoft** и включенная ею в состав операционной системы **Windows** (95, 98, NT).

Опишем кратко работу с указанной программой, широко используемой в настоящее время для навигации в сети Internet.

В окне **Internet Explorer** отображается HTML-документ, полученный из **Internet** или загруженный из файла. Навигация (перемещение) по такому документу осуществляется с помощью ссылок - выделенных в документе фраз или областей. Курсор превращается на ссылках в пиктограмму кисти руки с вытянутым пальцем, которым можно выбрать или нажать выделенную фразу или область. Полезные для навигации кнопки-команды содержатся в меню команд **Internet Explorer**- третьей строке окна. Кнопка «Назад» позволяет возвратиться на предыдущую просмотренную HTML-страницу, кнопка «Вперед» - двигаться вперед по цепочке просмотренных страниц. Кнопка «Останов» служит для остановки передачи данных (если, например, не нужно ждать окончательной досылки картинок и т.п.), кнопка «Обновить», напротив, позволяет возобновить передачу данных текущей страницы в случае, например, сбоя передачи или остановки.

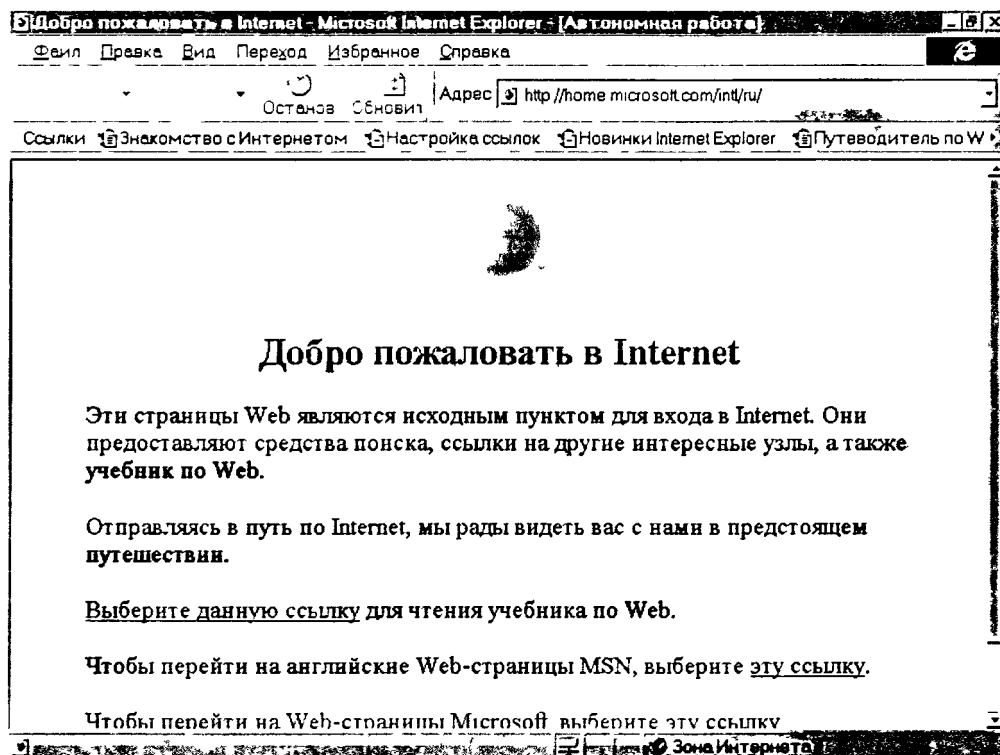


Рис. 5.8. Возможный вид заставки программы Internet Explorer

В той же строке есть очень важное окно ввода адреса. Обычно, в это окно вводят адрес домашней страницы WWW-сервера, с которой начинается поиск и просмотр всего HTML-документа. С помощью кнопки со стрелкой в правой части этого окна можно вызвать список ранее использовавшихся адресов и выбрать необходимый. Более широкие возможности «запоминания» адресов предоставляет опция меню «Избранное» в командном меню системы во второй строке. Эта опция открывает вертикальное меню, команды которого позволяют добавлять новые «закладки» на понравившиеся HTML-страницы, переходить в сети по этим закладкам, упорядочивать ссылки и даже работать с папкой «Мои документы».

Интереснейшая возможность, предусмотренная **Internet Explorer** - автоматическое установление связи с сервером компании **Microsoft** и обновление компонентов операционной системы (драйверов, приложений). Конечно, на такой сервис могут рассчитывать только владельцы лицензионных экземпляров системы.

Из команд главного меню **Internet Explorer** очень полезными являются также «Свойства обозревателя» в команде «Вид», позволяющая настраивать браузер, «Шрифты», позволяющая менять кодировку отображаемого в окне текста. Для кодировки русских символов в сети применяется как кодировка КОИ-8, так и **Windows**, встречаются национальные алфавиты. Команда «Шрифты» позволяет выбрать необходимую для правильного отображения текста кодировку.

В **Internet Explorer** нет встроенных средств разработки HTML-документов. Эта функция переложена на текстовый процессор **Word** из пакета **Microsoft Office**. **Internet Explorer** лишь вы-

зывает **Word**, когда необходимо, а также позволяет посылать подготовленные страницы на WWW-сервер (если у пользователя имеются соответствующие права).

В команде «Файл» предусмотрены возможности печати или записи на диск найденной информации.

Отметим, что до сих пор рассматривалась работа **Internet Explorer** с протоколом HTTP (т.е. WWW-серверами) по умолчанию. В действительности он, как и другие браузеры (например. Netscape Communicator), может работать и с другими протоколами Internet: FTP, GOPHER и т.д. Так, для соединения с любым FTP-сервером достаточно в поле адреса указать протокол и адрес сервера, например: **ftp://ftp.kiae.su**. Работать с файлами и директориями FTP-сервера в среде **Internet Explorer** можно в традиционной для **Windows** манере.

Технологии **Internet** быстро прогрессируют. Так, в едином пакете с **Internet Explorer** поставляется программное средство **Net Meeting** для участия в видеоконференциях по **Internet**.

3.4.5. Другие информационные системы в Internet

Gopher-это еще одна распределенная информационная система **Internet**. В основу ее интерфейсов положена идея иерархических каталогов. Внешне **Gopher** выглядит как огромная файловая система, которая расположена на машинах сети. Первоначально **Gopher** задумывался как информационная система университета с информационными ресурсами факультетов, кафедр, общежития и т.п. До сих пор основные информационные ресурсы **Gopher** сосредоточены в университетах. **Gopher** считается простой системой, легкой в установке и администрировании, достаточно надежной и защищенной. Количество серверов **Gopher** на 1994 г. превышало в 1,5 раза число серверов WWW, и до 1995 г. темпы роста установок серверов **Gopher** опережали все остальные ресурсы Сети. В России **Gopher**-серверы не так распространены, как во всем мире - профессионалам больше нравится **World Wide Web**. **Gopher URL's** не так разнообразны, как файловые. Это связано с ограниченностью его сервиса. Для того чтобы описать, например, **gopher-сервер** узла **gopher.kiae.su** необходим URL:

<gopher://gopher.kiae.su/>

WAIS - это распределенная информационно-поисковая система Internet. Родилась WAIS как перспективная разработка четырех ведущих американских компаний и первое время была коммерческим продуктом, пока не появилась ее свободно распространяемая версия freeWAIS. В основу системы положен принцип поиска информации с использованием логических запросов, основанных на применении ключевых слов. Клиент «обшаривает» все серверы WAIS на предмет наличия на них документов, удовлетворяющих запросу. WAIS широко применяется как поисковая машина в других информационных сервисах **Internet**, например в WWW и **Gopher**. Наиболее известным проектом, где была применена WAIS, является электронная версия энциклопедии «Британника».

Контрольные вопросы и задания

1. В чем признак глобальности сети?
2. Что такое «модем» и каковы его функции?
3. Какие модемы используются в Вашем вузе и какие протоколы коррекции ошибок они поддерживают?
4. В какой глобальной сети участвует ваш вуз? К каким сетям есть выход из этой сети?
5. Назовите характеристики одной - двух локальных сетей.
6. Какие виды обмена информацией приняты в электронной почте?
7. Какова структура электронного письма?
8. Каков электронный адрес вашего вуза и что означают его составляющие?
9. В чем состоят возможности одной из программ-оболочек электронной почты?
10. Для чего используется кодирование писем?
11. Приходилось ли вам лично работать с удаленной базой данных или наблюдать за такой работой? Если да, то опишите ее в деталях.

12. Какие виды работ поддерживает сеть Internet?
13. Что такое гипертекст? С какой гипертекстовой программой вы работали и как в ней отражены принципы гипертекста?
14. Как ведется поиск информации в сети Internet?

§ 4. ПРЕДСТАВЛЕНИЕ ОБ ОПЕРАЦИОННОЙ СИСТЕМЕ UNIX

ОС UNIX - одно из высших достижений в мире системного программного обеспечения. Многие специалисты считают, что по сравнению с ней MS DOS является гораздо менее профессиональной системой.

Хотя в специальных руководствах по UNIX подчеркивается ее простота и ясность, это не та простота, которая легко усваивается. Здесь мы познакомимся лишь с одним аспектом UNIX - управлением большой компьютерной сетью. Кроме того, эта ОС и тесно связанный с ней язык программирования Си являются основой для разработки сложных профессиональных программных продуктов самого разного назначения - как системных, так и прикладных.

Для глобальных сетей UNIX и UNIX-подобные системы (например, LINUX) являются основными. Здесь важно подчеркнуть, что UNIX прозрачным образом поддерживает не только работу с удаленного терминала (даже по телефонной линии), но и электронную почту, и набор протоколов TCP/IP. При этом детали обмена данными между компьютерами от пользователя системы скрыты, и он может, работая за любым компьютером сети или за удаленным терминалом, выполнять разнообразные операции и даже запускать процессы, не зная, где физически находится исполняющий компьютер.

Рассмотрим основные команды операционной системы UNIX. Сеанс работы с операционной системой UNIX начинается с ввода идентификатора пользователя и его пароля, присвоенных ему администратором сети при регистрации. Приглашением по вводу идентификатора является слово **login**. Пароль при вводе не отображается. По желанию пользователя он может быть заменен по команде **passwd**.

Командная строка - это последовательность слов, разделенных пробелами. Первое слово командной строки и есть команда, остальные - параметры.

Типы параметров:

- имя файла - идентификатор (использует символы a - z, A - Z, 0 - 9, _,.,-);
- опция (ключ) уточняет смысл команды (обычно начинается со знака «минус»); например, опция **-al** (может быть со знаком + или без него); смысл опции зависит от команды;
- выражение описывает обычно строку символов или является строкой.

Порядок параметров в команде:

1	2	3	4
command	options	expression	filename(s)

Система содержит подробное справочное руководство (UNIX Reference Manual), состоящее из восьми секций:

- 1) commands (команды);
- 2) system class (системные вызовы);
- 3) subroutines (подпрограммы);
- 4) special files (спецфайлы);
- 5) file format and convension (формат файлов и соглашения);
- 6) games (игры);
- 7) macro packages and language conventions (макропакеты и языковые соглашения);
- 8) administrator commands and procedures (команды и процедуры администратора).

В любой момент руководство может быть выведено на экран и принтер.

Описание команды состоит из следующих разделов:

- NAME (имя и функция);
- SYNOPSIS (синопсис);
- DESCRIPTION (описание функции);

FILE (используемые файлы);
SEE ALSO (смежные команды);
DIAGNOSTIC (реакция на ошибки);
BUGS (замеченные некорректности),

\$ man i <имя> (i - номер секции, например "\$ man grep"; если i опущено, подразумевается i=1).

Запуск программы (командного файла) из текущего каталога осуществляется при вводе имени содержащего ее файла в командной строке. Система обладает набором текстовых редакторов (например, VI и EX), интерпретатором команд **shell** и другими встроенными утилитами.

Узнать имя текущего каталога можно следующим образом:

\$ pwd (print working directory)

Для печати (вывода на экран) содержимого каталога используется команда **ls**.

Примеры.

\$ ls	- печать текущего каталога;
\$ ls-al	- а (all) - все файлы и подкаталоги;
	-l (long) - полную информацию;
\$ ls /	- печать каталога root ,
\$ ls -l /	- печать каталога root полная;
\$ ls -l /bin	- часто используемые команды;
\$ ls /dev	- драйверы устройств;
\$ ls /users	- каталоги пользователей.

Изменение рабочего каталога производится командой **cd** (change directory).

\$ cd /etc \$ ls -l	- печать команд администратора;
\$ cd /usr \$ ls -l bin	- редко используемые команды;
\$ cd - без параметров	- возврат к собственному (<i>fioine</i>) каталогу.

Команда **cat** (от concatenate) позволяет объединить несколько (два или более) файлов и направить результат на стандартный вывод (**Standart Output**), обычно - на экран дисплея, например:

\$ cat /etc/motd посмотреть файл (message of today)

Создать нижележащие каталоги можно командой **mkdir** (make directory). Копирование файлов осуществляется командой **cp**. Например, если текущим каталогом является **progs/c**, то ввод команды

\$ cp /etc/motd message

приводит к созданию в текущем каталоге **progs/c** файла **progs/c/mtssage**. Его можно было бы скопировать в текущий или в другой каталог, не меняя имени:

\$ cp /etc/motd progs/c

(при этом будет создан файл **progs/c/motd**). Если **progs/c** - текущий каталог, того же эффекта достигает команда

\$ cp /etc/motd

(с точкой в качестве параметра). Если второй аргумент команды **cp** - каталог, то в качестве первого можно указывать несколько файлов, например:

```
$ cp /etc/motd /usr/include/stdio.h progs/c
```

Перенос и переименование файлов осуществляется командой **mv**. Эта команда перемещает файл или каталог из одного места файловой системы в другое. Побочный эффект - переименование файла.

Пример.

```
$ mv message mesg
```

```
$ls
```

```
mesg....
```

Отличие от копирования состоит в том, что исходный файл будет уничтожен. Удаление (уничтожение) файлов производится с помощью команды **rm**.

Пример:

```
$ cd progs/c
```

```
$ rm motd
```

```
$ ls message (нет motd)
```

Одной командой **rm** можно удалить несколько файлов.

Ключ **-i** позволяет использовать интерактивный вариант исполнения команды, требующий подтверждения на удаление каждого файла:

```
$ rm-i file1 file2
```

```
file1 : n (no-нет)
```

```
file2: y (yes-да)
```

Можно форсировать уничтожение ключом **-f**, даже если файл защищен по записи, например:

```
$ rm-fstdio.h
```

Этимися средствами нельзя уничтожить каталог, пока не уничтожены все файлы, входящие в его состав.

Для удаления нескольких каталогов используется ключ рекурсивного уничтожения **-r** (уничтожить все поддиректории). Пустой каталог удаляется посредством команды **rmdir** (remove directory):

```
$ rmdir progs
```

Метасимволы служат для подстановки любых строк и символов в именах файлов и в командах языка задания **Shell**:

* - представляет произвольную строку (возможно пустую);

? - любой одиночный знак;

[C1 C2] - любая литера из диапазона C1- C2 (в стандарте ASCII).

Примеры:

1) \$ ls c?

```
c1 c2 c3 cs cz
```

2) \$ ls c*

```
c1 c12 c2 c23 c3 cs csl cxy cz
```

3) \$ ls ?l*

```
c1c12
```

Для перенаправления вывода используется знак ">".

Пример.

```
$ ls -l - вывод на экран = standard output;
```

```
$ ls -l > dirconts - вывод в файл dirconts;
```

```
$ cat dirconts - вывод на экран.
```

Пробелы вокруг символа «>» необязательны. Возможно перенаправление вывода с добавлением (с дописыванием в файл), которое обозначается «>>».

Пример.

\$ pwd >> dirconts - добавить в файл имя текущего каталога.

Для перенаправления стандартного ввода используется знак «<».

Пример.

\$ mail - ввод сообщения с экрана;

\$ mail < message - ввод сообщения из файла **message**.

Эта возможность используется реже, чем перенаправление вывода.

Конвейером называется группа команд, объединенных программными каналами. Программный канал образуется назначением стандартного вывода одной команды стандартным вводом следующей команды. Для формирования программного канала используется знак «|» («вертикальная черта»).

Система UNIX располагает развитыми средствами связи пользователей между собой. К ним относятся команды:

- **who** - позволяющая выяснить, кто работает в данный момент с системой;
- **write** - для непосредственной отправки сообщения прямо на экран терминала другому пользователю;
- **mail** - реализующая электронную почту, использующую электронные почтовые ящики.

Команда **write** посылает сообщение указанному пользователю в момент исполнения этой команды:

write <пользователь адресат><сообщение>

Сообщение заканчивается выводом знака конца файла **control-D**. Если ожидается ответ и предполагается диалог можно не вводить знак конца файла до конца диалога. Электронная почта позволяет пользователям обмениваться сообщениями, которые попадают в почтовые ящики, реализованные внутри системы. При входе пользователя в систему появляется извещение о наличии для него почты, например:

login: password:... you have mail \$

В отличие от команды **write**, в системе **mail** о приходе почты пользователя известят, когда он закончит текущую работу. Пост;» пит сообщение:

you have mail (для вас есть почта).

Для получения почты необходимо ввести команду **mail** без параметров:

\$ mail

Подсказка (?) означает, что система **mail** ждет оказания о том, что делать с почтой: печатать, сохранить, уничтожить или выйти из команды **mail**. Ввод знака вопроса приводит к меню возможных действий:

q quit	- закончить;
x exit without changing mail	- выйти без изменения почты;
p print	- печать;
s [file] save	- сохранить в файле;
w [file] same without header	- то же без заголовка;
- print previous	- печатать предыдущее;
d delete	- уничтожить;
+ next (no delete)	- следующее (не уничтожать);
m user mail to user	- переправить другому пользователю;
!cmd execute cmd	- выполнить команду;
?	

Для отправки почты следует ввести команду **mail** с параметром-именем пользователя и

текст письма, заканчивая его знаком конца файла (**control-D**). Для выхода из системы mail нужно нажать q, например:

? q you have mail \$

Существуют и более развитые средства работы с электронной почтой, например утилита **bmail**. Чрезвычайно важно, что те же средства позволяют обмениваться электронной почтой не только в рамках локальной сети, но и глобально, в рамках **Internet**. Для этого необходимо лишь указывать идентификатор адресата с интернетовским адресом его локальной сети. Сама передача сообщений по каналам связи ведется в этом случае с помощью средства под названием UUCP (Unix to Unix copy).

Помимо перечисленных команд UNIX имеет развитые средства обработки текстов, встроенные компиляторы с языков программирования Си, Фортран77, Паскаль и др, а также средства разработки программ.

Следующий уровень взаимодействия пользователей в сетях достигается на основе совместного использования распределенных по сети файлов. Прозрачным использованием таких файлов делают сетевая файловая система NFS (Network File System) и сетевая информационная служба NIS (Network Information Service). NFS использует модель клиент/сервер, в которой сервер экспортирует доступные для совместного использования каталоги, а клиент монтирует эти каталоги, чтобы обращаться к хранящимся в них файлам. До сих пор мы рассматривали команды UNIX, выполняемые из командной строки. Для обеспечения выполнения части из них, таких как перемещение по каталогам и их создание, копирование файлов и запуск программ, служит оболочка UNIX под названием **Demos Commander**, в основных чертах аналогичная **Norton Commander**. Аналогом графической среды **Windows** в UNIX является графическая оболочка **X Windows**.

Контрольные вопросы и задания

1. Охарактеризуйте основные особенности и команды ОС UNIX.
2. Опишите основные команды ОС UNIX.
3. Какова роль операционной системы UNIX в обеспечении сервиса Internet?

§ 5. ИСПОЛЬЗОВАНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ В ОБРАЗОВАНИИ

5.1. ТЕЛЕКОММУНИКАЦИИ КАК СРЕДСТВО ОБРАЗОВАТЕЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Компьютерные телекоммуникации все настойчивее проникают в различные сферы жизни современного общества: бизнес, финансы, средства массовой информации, науку и образование.

Несмотря на то что Россия занимает сейчас 37-е место в мире по уровню телефонизации (являющейся одним из важнейших факторов, обуславливающих развитие компьютерных телекоммуникаций в стране), за последние 2-3 года российские пользователи персональных ЭВМ получили доступ ко множеству зарубежных телекоммуникационных сетей.

На общем фоне развития телекоммуникаций в нашей стране постепенно проявляется и становится заметным процесс внедрения компьютерных телекоммуникаций в сферу народного образования, и, прежде всего, в жизнь современной школы. Десятки тысяч школ за рубежом и сотни школ в нашей стране за последние 5-7 лет начали использовать возможности современных телекоммуникаций непосредственно в учебной работе.

Особенно стремятся участвовать в этом процессе школы из «глубинки», где уже есть современная телефонная сеть, но по-прежнему нет доступа к современной оперативной информации по различным отраслям знаний.

Некоторые учителя используют телекоммуникации преимущественно для внеклассной работы с учащимися по отдельным экспериментальным проектам. Однако уже сейчас многие школы за рубежом используют компьютерные телекоммуникации непосредственно на уроках в условиях реального учебного процесса, постепенно подготавливая учащихся к жизни в информационном обществе. Компьютерные телекоммуникации начинают постепенно осознаваться многими педаго-

гами как один из инструментов познания окружающего мира. Инструмент этот настолько мощный, что вместе с ним в школу приходят новые формы и методы обучения, новая идеология глобального мышления.

Компьютерные телекоммуникации - интенсивно развивающийся вид информационных технологий, использующий глобальные компьютерные сети, - обещают совершить переворот в методах и формах обучения. Простейший вид телекоммуникаций - электронная почта - уже сейчас, с минимальными затратами, с успехом может быть использован в учебном процессе каждой школы. Учебное значение электронной почты состоит в том, что она

- стимулирует и облегчает обмен опытом преподавателей различных предметов;
- повышает интерес учащихся к учебному курсу, в котором используется;
- расширяет коммуникативную практику учащихся, помогает в совершенствовании письменной речи;
- делает возможным использование новых методических приемов, основанных на сопоставлении собственных данных учащихся и тех, которые получены по электронной почте.

Использование электронной почты в обучении обычно протекает в форме телекоммуникационных проектов. Учебный телекоммуникационный проект посвящается определенной теме, включает разнообразные виды деятельности учащихся по подготовке и передаче, а также получению и анализу учебной информации с помощью средств компьютерных телекоммуникации, и охватывает по времени от нескольких дней до нескольких месяцев.

Простые телекоммуникационные проекты могут выполняться двумя классами учащихся под непосредственным руководством учителей и проходят в форме неструктурированной коллективной переписки. Сложные длительные проекты, в которых участвуют десятки и сотни классов, требуют участия в проектах координаторов и методистов, руководящих ходом телекоммуникаций, добивающихся согласованности содержания и сроков отправки корреспонденции. Большие проекты проводятся в специализированных учебных компьютерных сетях (США, Канады, Великобритании и т.д.).

В России учебные компьютерные сети еще не развиты, и в настоящий момент российское образование делает первые шаги к учебному использованию компьютерной электронной почты как средству управления учебной мотивацией. Публикация с помощью электронной почты оказывается оперативной целью овладения компьютером, проведения наблюдений и измерений, коллективной деятельности учащихся.

Во многих школах России имеется опыт следующего использования электронной почты:

- для обмена опытом педагогов, проводящих различные учебные курсы;
- для обмена между учащимися коллективными сообщениями следующих **типов**:
 - письма общего характера (письмо-представление, рассказ о своем городе, о школьных друзьях и о спортивных достижениях, рассказ о домашних животных, поздравления с праздниками и др.);
 - писем-отчетов о проведенных исследованиях и выполненных проектах;
 - датаграфических и фенологических наблюдений за погодой;
- для проведения компьютерных телекоммуникационных викторин;
- для «распределенного» выполнения исследований (лабораторные исследования и измерения выполняет один из классов, а остальные классы обобщают присланные им данные измерений и результаты наблюдений).

В образовательной практике могут найти применение все основные виды компьютерных телекоммуникаций:

- электронная почта;
- списки рассылки (list-серверы);
- электронные доски объявлений (BBS);
- телеконференции;
- **Gopher**- и WWW-серверы.

Там же находят применения различные информационные технологии общения, в частности:

- общение многих со многими;
- общение, не ограниченное географическими расстояниями;
- общение, не ограниченное временными рамками;

- общение на основе текста, а не речи.

Установлено, что общение, опосредованное компьютером, влияет на основные характеристики общения людей с помощью компьютерных сетей.

Комплексное использование информационных и коммуникативных возможностей *Internet* обладает очень большими потенциальными возможностями в образовании. Возможности этой системы для поддержки свободного обмена информацией поразительны и практически безграничны. Рассмотрим основные идеи использования телекоммуникационных средств в образовательном контексте, основанные на классификации большого количества файлов, полученных в результате работы в *Internet*. Данный материал может помочь спланировать эффективное использование телекоммуникаций в обучении, полностью интегрированное с изучаемым учебным курсом.

5.2. ПЕРСОНАЛЬНЫЙ ОБМЕН СООБЩЕНИЯМИ

Наиболее популярный тип телекоммуникаций состоит в электронном общении индивидов с другими индивидами, индивидов с группами, групп с группами. Многие из телекоммуникационных проектов используют электронную почту (иногда посредством дискуссионных групп на list-серверах) как единую среду общения. В других проектах преподаватели и учащиеся используют телеконференции и связанные с *Internet* «доски объявлений». Можно выделить следующие основные разновидности телекоммуникационных проектов.

Свободная переписка. Возможно, это наиболее часто используемая структура телекоммуникационной деятельности, аналогичная переписке по обычной почте. Поскольку переписка между учащимися требует большего внимания, чем многие преподаватели могут ей уделить, можно развернуть свободный обмен электронной почтой между группами. Свободная переписка является прекрасным источником при изучении иностранных языков, культурных традиций различных народов и т.д.

Вот пример вводного сообщения от группы французских учащихся, желающих узнать о том, как в других странах изучаются компьютерные технологии :

«Здравствуйте! Мы -группа 13-16-летних учащихся технологической школы им. Ромена Роллана в пригороде Парижа. На занятиях по технологиям ученики должны разработать проект, связанный с технологиями и компьютерами. Мы хотим переписываться с вами, чтобы узнать о вас больше:

- Что представляет собой ваш класс?
- Где находится ваша школа?
- Изучаете ли вы технологии?
- Что вам больше всего нравится делать?

Мы хотели бы также переписываться о различных компьютерных играх».

Глобальный класс. При этой разновидности структуры переписки два или более класса (находящиеся где угодно) могут совместно изучать одну и ту же тему, обсуждая то, что они сейчас изучают по заранее определенному графику.

Вот пример «электронного расписания» месячных проектов по ознакомлению с опасностью СПИДа :

- 1-15 февраля. Регистрация.
- 1- 5 марта. Что представляет собой вирус СПИДа?
- 8-12 марта. СПИД и образование.
- 15- 19 марта. Выявление СПИДа.
- 22 - 26 марта. Обмен мнениями о СПИДе.

Предложения для обсуждения и отработки на весь срок проекта включаются в план. Хорошо бы вывесить на стенд сообщения от учащихся, преподавателей и ученых на тему проекта. Побуждайте своих учащихся готовить сообщения на эту тему или на любую другую, которую они хотели бы обсудить. Эти сообщения должны быть красиво оформлены на специальном стенде, чтобы их могли видеть другие учащиеся, родители, администрация, посетители. Учащиеся и учителя участвуют в обсуждении темы через электронную почту.

Электронные «встречи». Электронная почта, телеконференции, электронные доски объявлений могут также дополняться синхронным общением в реальном времени. При этом общение между учащимися и их «гостем» происходит путем поочередной печати на клавиатуре с использо-

ванием возможности «переговоров», допускаемой многими системами электронной почты.

Электронное обучение. Специалисты различных профилей из вузов, частных и государственных учреждений, предприятий или школ, связанные через **Internet**, могут служить «электронными преподавателями» для учащихся, желающих познакомиться со специальными темами в интерактивном режиме. Так, например, в учебных целях может быть организовано общение учащихся с руководителями, политиками, писателями и другими экспертами в определенной сфере деятельности. Другая разновидность такого рода проекта - когда в качестве «электронных педагогов» выступают студенты старших курсов некоторых вузов по отношению к школьникам в других городах и даже странах. Получила известность поисковая служба под названием «Электронные Эмиссары», спонсируемая Техасским центром образовательных технологий и Техасским университетом в Остине. Она помогает желающим найти и установить контакты с экспертами в различных областях знаний во всех странах мира. С ее помощью преподаватели и их классы находят друг друга, разрабатывают структуру учебного проекта и обучаются посредством электронной почты.

Ролевые игры. В проектах, основанных на ролевых играх, участники общаются друг с другом, играя определенную роль. К примеру, профессор истории университета Вирджинии Дженнингс Ваггонер становится Томасом Джефферсоном для нескольких классов средних школ, изучающих историю. В проекте, спонсируемом госдепартаментом образования штата Небраска и университетом Небраски в Омахе, студенты старших курсов по электронной почте играют роли главных персонажей книг, которые учащиеся средних школ Восточной Небраски изучают со своими учителями. Существует большое число таких проектов исторической направленности.

5.3. ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ

Некоторые из наиболее успешных образовательных телекоммуникационных проектов построены на сборе, обработке, сопоставлении учащимися различного рода информации.

Информационный обмен. Имеется много примеров тематического обмена информацией, которые могут рассматриваться как телекоммуникационные проекты.

Учащиеся всего мира и их учителя собирают

- народные игры;
- жаргонные слова;
- шутки;
- пословицы;
- народные сказки;
- местную сельскохозяйственную информацию;
- информацию о здоровье;
- описания местных и национальных праздников;
- афоризмы;
- туристическую информацию о городах.

В проектах этого типа может участвовать много классов и управление ими не становится чрезмерно сложной задачей для учителей. Это весьма плодотворное приложение телекоммуникационных средств, поскольку дети оказываются одновременно создателями и потребителями информации, которой они обмениваются. Проекты такого типа обычно начинаются с приглашения к участию, которое рассылается учителем одного из классов.

Электронные публикации. Другой вид сбора и обмена информацией связан с электронной подготовкой и публикацией общего труда, такого как газета, альманах или литературный журнал. Имеется много примеров таких проектов. Так, известен проект совместного написания «Всемирной поэмы мира». Такая электронная поэма несколько раз обходит весь земной шар от класса к классу, которые добавляют к ней по куплету, прочитав ранее созданное другими классами.

Создание базы данных. Некоторые проекты обмена информацией используют не только сбор информации, но и ее организацию в виде базы данных, которую участники проекта и другие учащиеся могут использовать для обучения. Примером такого проекта является проект «Времена года», имеющий целью привести учащихся к пониманию причин смены времен года и их особенностей в различных климатических зонах. В ходе проекта учащиеся собирают данные и создают базу данных, содержащую информацию о погоде. Отметим, что удачные проекты этого типа хо-

рошо структурированы; они имеют определенное расписание, ясно сформулированные условия участия, побуждают учителей (часто путем заполнения регистрационных форм) следовать определенным руководствам.

Телекоммуникационные экскурсии. Ряд телекоммуникационных проектов, проводимых в *Internet*, построен на обмене учителями и учениками наблюдениями и впечатлениями от экскурсий в музеи, исторические места, парки, зоопарки и т.д. с учителями и учащимися из других мест, городов и стран. В *Internet* составляется месячное расписание передачи информации об экскурсиях, пересылаемой из школ, и направляется заинтересованным учителям. Если экскурсия дала информацию, полезную для каких-либо уроков, можно в соответствии с расписанием задать вопросы детям, совершившим эту экскурсию,

Путешествия и экспедиции, совершаемые специалистами, также обсуждаются в *Internet*. Так, международный арктический поход через Северный Ледовитый Океан на собачьих упряжках и каноэ описывался учителями - участниками экспедиции - через list-сервер. Во время недавней экспедиции, предпринятой двумя исследователями из Великобритании, классы-участники еженедельно получали детальное описание хода путешествия, наблюдений, впечатлений. По возвращении путешественников в Великобританию их ждало множество электронных писем с вопросами от детей со всего мира.

Совместный анализ данных (в том числе обзоры). Информационный обмен особенно плодотворен, когда данные собираются в различных местах, а затем подвергаются сопоставлению и/или числовому анализу. Простейшие типы таких проектов привлекают учащихся к электронной подготовке обзоров, сбору данных, анализу результатов, отчетов о том, что было обнаружено. Проекты по совместному сбору данных также могут включать:

- проекты по определению кислотности воды (дождевая или грунтовая вода собирается на различных участках, проверяется на кислотность, а затем проводится анализ, в каком месте и когда вода была наиболее загрязненной);
- проекты, проводимые несколькими различными организациями в месяцы, предшествующие выборам; в этих проектах учащиеся голосуют электронным способом, а затем результаты сопоставляются с реальными результатами голосования;
- проекты определения всемирной потребительской корзины (учащиеся сравнивают цены стандартного набора товаров в различных регионах планеты, а затем пытаются обосновать причины различия цен);
- телекоммуникационные олимпиады (учащиеся разных школ проводят соревнования типа олимпийских игр, а затем сопоставляют результаты соревнований, для того чтобы определить победителей каждого виртуального состязания);
- эксперимент Эратосфена (учащиеся со всего мира повторяют геометрическую процедуру Эратосфена по оценке длины окружности Земли, измеряя угол тени вертикально установленной на земле палочки в полдень осеннего (или весеннего) равноденствия);
- проект по определению миграции бабочек одного вида (наблюдается и прослеживается миграция бабочек с помощью их маркировки и отбора экземпляров из групп мигрирующих бабочек).

Очевидно, что этот тип проектов является очень многообещающим с точки зрения привлечения учащихся к крупномасштабным исследованиям для получения ответов на сложные и интересные вопросы.

5.4. СОВМЕСТНОЕ РЕШЕНИЕ ЗАДАЧ

Совместное решение задач может оказаться новым, очень перспективным наполнением образовательной телекоммуникационной среды. Проекты могут быть основаны как на соревновании, так и на сотрудничестве, но примеры существующих проектов показывают, что учителя и учащиеся предпочитают сотрудничество.

Поиск информации. В *on-line* проектах этого типа учащиеся должны использовать различные источники информации (электронные или бумажные) для решения задач. Им дается также ключ к решению. Известна географическая игра, в которой каждой из 20 участвующих групп учащихся дается однотипный набор информации о месте размещения их школы (широта, долгота, часовая пояс, численность населения, расстояние до столицы страны и т.д.). Координаторы игры пе-

ремешивают эти данные, а все учащиеся, используя справочную информацию (карты, атласы, географические справочники и т.д.) находят города и ставят им в соответствие информацию. Побеждает тот класс, который установит наиболее точное соответствие.

Электронное сочинение. Известны канадские проекты, в которых классы регулярно посылают сочинения в телеконференции. Другие учащиеся могут следить за работой такой электронной литературной секции. К проекту привлекаются профессиональные писатели, которые дают свои критические конструктивные заметки, делятся секретами мастерства, публикуют отрывки из своих произведений, над которыми они работают.

Одновременное выполнение заданий (в том числе конкурсы). В этом типе проектов учащимся, находящимся в разных местах, предлагают одинаковые задания для их выполнения. Затем происходит электронный обмен решениями. Получили известность архитектурные проекты такого, например, типа: «Какую самую высокую конструкцию вы сможете построить из спичек, чтобы она

- 1) могла нести на себе яйцо;
- 2) выдерживала, если на нее сильно подуют?»

При этом можно использовать только определенный клей. Яйцо должно быть сварено вкрутую и не очищено. Для оценки высылаются описание и рисунок.

Моделирование. On-line моделирование - это такие телекоммуникационные проекты, которые требуют, возможно, наибольшей координации и поддержки,. Однако эффективность обучения и увлеченность участников вполне оправдывают дополнительные затраты времени и усилий со стороны организаторов проекта. Среди успешных проектов **on-line** моделирования назовем следующие.

Моделируется космическая программа «Шаттл». Школы всего мира играют различные роли в моделируемом полете космического корабля «Шаттл». Исследование включает маневр стыковки с другим «Шаттлом», управление полетом с наземного пульта управления, приземление в различных местах, наблюдение обсерватории солнечных возмущений и т.д. Координация и коммуникации между пультом управления «Шаттла» и другими школами осуществляется через распределенную конференцию на индивидуальных интерактивных сетевых системах. Электронная почта пересылается очень быстро, ежечасно публикуются отчеты, даже возможны электронные переговоры в реальном масштабе времени между пультом управления, астронавтами и вспомогательными подразделениями.

Известна прекрасная серия проектов моделирования международных событий и проблем, а также разрешения глобальных конфликтов. Упомянем проект IDEALS, основанный на моделировании учащимися деятельности и поведения высокопоставленных дипломатов, представляющих различные страны на международной конференции. Они должны, например, выработать текст договора об использовании океанских ресурсов или о будущем Антарктиды.

Социальные проекты. У граждан мира. живущих в конце XX века, не должно вызывать удивления то, что **Internet** служит средой для гуманитарных межкультурных телекоммуникационных проектов, ориентированных на деятельность, в которых участвуют будущие руководители планеты - наши дети. Так, проект **Planet** использует консорциум шести связанных с **Internet** образовательных сетей, представители которых работают вместе над созданием проектов полезной, важной общественной деятельности, в которой дети впервые ощущают ответственность за решение глобальных проблем огромной важности.

В течение первых месяцев работы проекта **Planet** его участники писали в ООН петиции протеста против событий в Югославии, методом мозгового штурма разрешали политическую ситуацию в Сомали, планировали и искали источники финансирования орошения засушливых местностей Никарагуа. Потенциал таких проектов для мультидисциплинарного, устремленного в будущее, построенного на истинном сотрудничестве обучения несомненен.

К сожалению, большинство учебных компьютерных проектов, функционирующих в **Internet** и разработанных в США, в России неизвестно. Некоторую известность у нас получили учебные проекты сети KIDNET. Недостатком такого типа проектов является их ориентация на традиции, сложившиеся в образовании США: прагматичность, поверхностность, бессистемность. Такие проекты с трудом вписываются в предметную систему российского образования.

В ближайшие годы наилучшими перспективами обладают такие технологии обучения с использованием компьютерных телекоммуникаций, которые основаны на телекоммуникационной

поддержке традиционных форм обучения и носят характер коллективно выполняемых под руководством педагога учебных проектов.

Такой характер имеют, в частности, получившие известность российские телекоммуникационные образовательные проекты, ориентированные на среднюю школу, такие как телекоммуникационная олимпиада по программированию, проведенная в Переславле-Залесском, олимпиада по физике, поддерживаемая в Красноярске, экономические игры МЭМ (моделирование экономики и менеджмента), методическое обеспечение которых выполняется в КУДИЦ (Москва), телекоммуникационные викторины (олимпиады) по ряду школьных предметов, экономическая бесконкурентная игра «экономические связи», проект изучения английского языка в переписке с американскими школьниками, методика которых разработана в Воронежском госпедуниверситете.

Очевидно, что на начальных этапах внедрения образовательных технологий на основе компьютерных телекоммуникаций могут возникнуть существенные трудности и помехи, среди которых

- недостаточно насыщенный компьютерный парк учебных учреждений и индивидуальных пользователей,
- недостаточное развитие компьютерных телекоммуникационных сетей в России, их нестабильность,
- недостаточная компьютерная грамотность и информационная культура населения, что создает дополнительные психологические барьеры в развитии телекоммуникационных методов обучения.

Однако главной проблемой развития телекоммуникационного обучения является создание новых методов и технологий обучения, отвечающих телекоммуникационной среде общения. В этой среде ярко проявляется то обстоятельство, что учащиеся не просто пассивно потребляют информацию, в процессе обучения они создают собственное понимание предметного содержания обучения. В настоящее время требуется коррекция устаревшей модели обучения, которая характеризовалась тем, что

- в центре технологии обучения - учитель;
- между учащимися идет негласное соревнование;
- учащиеся играют пассивную роль на занятиях;
- суть обучения - передача знаний (фактов).

На смену устаревшей модели должна прийти новая модель обучения, основанная на следующих положениях:

- в центре технологии обучения - учащийся;
- в основе учебной деятельности - сотрудничество;
- учащиеся играют активную роль в обучении;
- суть технологии - развитие способности к самообучению. Основные группы задач, решаемых с помощью сети, включают в себя
 - поддержку учебной работы учащихся;
 - обеспечение взаимодействия между педагогами, обмен педагогическим опытом и дидактическими материалами;
 - обеспечение доступа всех участников учебно-воспитательного процесса к быстро растущим информационным фондам, хранящимся в централизованных информационных системах;
 - информационное обеспечение решения задач управления.

Контрольные вопросы и задания

1. Каковы перспективы использования компьютерных сетей в образовании?
2. В чем состоит учебное значение электронной почты?
3. Что называется учебным телекоммуникационным проектом?
4. Охарактеризуйте основные виды учебного использования Internet.
5. С какими проблемами сталкивается учебное применение телекоммуникаций в России?
6. Предложите идею своего собственного телекоммуникационного проекта.

Дополнительная литература к главе 5

1. *ВеттисД.* Novell Netware.-Киев: ВНУ, М., Бином, 1994. -
2. Вычислительные комплексы, системы и сети. /А.М.Ларионов, С.А.Майоров, Г.И. Новиков: Учеб. для вузов.-Л.: Энергоатомиздат, 1987.
3. Глобальные сети: информация и средства доступа. - Пермь: ПГТУ, 1994.
4. *Гольц Г.* Рабочие станции и информационные сети. - М.: Машиностроение, 1990.
5. *Дансмур М., Дейвис Г.* Операционная система UNIX и программирование на языке Си. М.: - Радио и связь, 1989.
6. Интернет. Всемирная компьютерная сеть. Практическое пособие и путеводитель. - М.: Синтез, 1995.
7. *Каракозов С. Д.* Введение в компьютерные сети. - Барнаул, БГПУ, 1996.
8. Компьютерные сети. Учебный курс Microsoft Corporation/ Пер. с англ. - М.: Издательский отдел «Русская Редакция» ТОО «Channel Trading Ltd.». - 1997.
9. *Крол Эд.* Все об INTERNET. Руководство и каталог. - Киев: ВНУ, 1995.
10. *Ларионов А.М., Майоров С. А., Новиков Г. И.* Вычислительные комплексы, системы, сети. - М.: Энергоатомиздат, 1987.
11. *Морозов В. К, Долганов А. В.* Основы теории информационных сетей. - М.: Высшая школа, 1987.
12. *Нанс Бэрри.* Компьютерные сети. - М.: БИНОМ, 1996.
13. *Рассохин Д., Лебедев А.* World Wide Web - всемирная информационная паутина в сети Internet. - М.: МГУ, 1997.
14. *Самойленко С. И.* Сети ЭВМ. - М.: Наука, 1986.
15. *Свириденко С. С.* Современные информационные технологии. - М.: Радио и связь, 1989.
16. Сетевые средства Windows NT. - С.-Пб.: ВНУ-Санкт-Петербург, 1996.
17. Стандарты по локальным вычислительным сетям. Справочник / Под ред. С.И.Самойленко. - М.: Радио и связь, 1990.
18. Телекоммуникационная сеть в системе образования Пермской области. Информационно-справочное пособие. Часть 1. - Пермь: ОЦПИ, 1995.
19. Телекоммуникация и информатизация общества. Серия «Информация. Наука. Общество». - М.: ИНИОН, 1990.
20. *Уваров А. Ю.* Организация и проведение учебных телекоммуникационных проектов. - Барнаул, БГПУ, 1996.
21. *Уваров А.Ю.* Учебные телекоммуникационные проекты в классе. - Барнаул, БГПУ, 1996.
22. *Хромцов П.* Лабиринты Internet. - М.: Электронинформ. 1996
23. Электронная почта в системе MS-DOS. Официальное руководство компании Редком. 1995г.
24. Журнал «Компьютерик-Москва», с 1995 г.

ГЛАВА 6

ИНФОРМАЦИОННЫЕ СИСТЕМЫ

ВВЕДЕНИЕ

В узком смысле, следуя традиции, под информационными понимают системы, предназначенные для хранения информации в специальном образом организованной форме и обеспечивающие ее ввод и различные манипуляции с ней, включая поиск по некоторым признакам, подготовку сводок (отчетов) и т.д.

Сама идея информационных систем и некоторые принципы их организации возникли задолго до появления ЭВМ. Библиотеки, архивы, адресные бюро, телефонные справочники, словари - все это информационные системы.

В современной информатике рассматриваются такие информационные системы, которые используют для хранения и обработки информации компьютерную технику. Именно компьютеризация придала информационным системам их современный облик, на несколько порядков повысила эффективность и расширила сферу их применения.

К информационным системам относят, в частности,

- информационно-справочные и информационно-поисковые системы;
- системы, обеспечивающие автоматизацию документооборота и учета, в том числе бухгалтерского;
- автоматизированные системы управления;
- экспертные системы;
- системы автоматизации научных исследований и автоматизированного проектирования, геоинформационные системы.

В последнее десятилетие большое развитие получил представляющий для нас особый интерес класс программных средств, которые также можно отнести к информационным - обучающие (информационно-обучающие) системы. Они также будут рассмотрены в этой главе.

Общей функцией всех информационных систем является сбор, накопление и хранение информации, ее обработка (в частности, поиск) и выдача в той или иной форме. Важнейшими подсистемами информационных систем являются банки информации. Ведущее направление развития информационных систем - совершенствование их интеллектуальных функций, облегчающих работу с ними, а также их сопровождение и развитие. Такими интеллектуальными функциями являются

- интерфейс с информационной системой на естественном языке (речевой ввод и вывод информации, формирование запросов на естественном языке);
- поддержка принятия решений, решения задач на основе информации, имеющейся в информационной системе.

В предыдущих главах мы остановились на таких аспектах представления и обработки данных с помощью компьютера, как информационное моделирование, базы данных и СУБД, языки управления реляционными базами данных, понятия и системы искусственного интеллекта, а также кибернетические аспекты информатики. Эти вопросы часто относят к единому разделу информатики под названием «Информационные системы»; туда же можно отнести и часть функций локальных и глобальных сетей. Таким образом, ниже внимание будет сосредоточено лишь на тех аспектах информационных систем, которые не обсуждались ранее.

§ 1. БАНКИ ИНФОРМАЦИИ

1.1. БАНКИ ДАННЫХ

Банк данных - наиболее характерный пример информационной системы. В банке данных хранится достаточно универсальная, необходимая для решения разнообразных прикладных задач, информация об определенной предметной области в специальном представлении, чаще всего предполагающем хранение и обработку с помощью компьютеров. При этом сами данные образуют базу данных, а банк, наряду с базой, содержит программные средства обработки данных и реализации запросов, т.е. систему управления базой данных (СУБД). Как правило, банки данных являются системами коллективного пользования. К информации, хранимой в них, часто можно получить доступ по телекоммуникационным сетям.

В современном мире существует огромное число банков данных. В них содержатся сведения коммерческого характера, данные по библиотечным фондам, системам здравоохранения, транспорта и т.д. Быстро развиваются банки, содержащие сведения о системах образования - национальных, региональных (в России они называются «банки педагогической информации»).

Итак, основа банка - база данных. Определение базы данных, основные понятия, связанные с различными моделями данных, уже обсуждались в гл. 2 и это обсуждение будет продолжено ниже в этой главе.

Остановимся на классификации банков информации. Эта классификация может быть проведена с разных точек зрения. По **назначению** можно выделить следующие классы банков информации:

- *информационно-справочные* системы (общего назначения и специализированные);
- банки данных в *автоматизированных системах управления* (предприятий и организаций, технологическими процессами и т.д.);
- банки данных в *системах автоматизации научных исследований*.

Однако такая классификация является не вполне строгой и завершенной. По **режиму**

функционирования можно рассматривать банки информации *пакетного, диалогового и смешанного* типов, В связи с широким распространением персональных компьютеров, локальных и глобальных сетей ЭВМ подавляющее распространение получили диалоговые системы.

По **архитектуре** вычислительной среды различают *централизованные и распределенные* банки информации.

К настоящему времени сложились следующие три основных типа банков информации: банки документов, банки данных и банки знаний.

Исторически первым типом банков информации явились банки документов или документальные информационно-поисковые системы. Документальные информационно-поисковые системы бурно развивались в 60-е годы, они широко используются в качестве справочного инструмента пользователей научно-технической информацией, в информационном обслуживании управленческих работников, специалистов и др. В настоящее время интерес к этим системам возобновился в связи с развитием глобальных информационных сетей (**Internet**) и появлением гипертекстовых серверов типа WWW, **Gopher** и т.д., которые вместе с соответствующими поисковыми системами {**Archie**, **Whatis** и т.д.) можно отнести к распределенным банкам документов.

Объектом хранения в таких системах является документ (научная статья, монография, приказ, циркуляр, письмо и т.д.) или факты, извлеченные из документов. Для обеспечения поиска и доступа к таким документам необходима их предварительная семантическая обработка - индексация. Индексация до настоящего времени остается неавтоматизированной процедурой и выполняется специалистами -людьми, индексирующими документы и запросы.

1.2. БАНКИ ДОКУМЕНТОВ

В документальной информационно-поисковой системе выделяют следующие компоненты:

- массив документов (текстов) или фактов, выступающих в качестве объектов хранения и поиска;
- информационно-поисковый язык, предназначенный для отображения содержания документов и операций над ними, в том числе и запросов для поиска документов;
- правила, алгоритмы, методы индексирования и поиска документов, позволяющие описывать документы и операция над ними на информационно-поисковом языке;
- комплекс программных и аппаратных средств, с помощью которых реализуются процессы накопления, хранения и поиска документов;
- обслуживающий персонал, включающий администратора банка документов, системных аналитиков, программистов и индексаторов. Банки документов работают обычно в двух режимах:

1) избирательного распределения информации, обеспечивающего регулярное текущее информирование пользователей банка о новых поступлениях документов;

2) ретроспективного поиска информации по разовым запросам во всем массиве документов.

Важнейший этап обработки нового документа при поступлении его в хранилище документальной информационно-поисковой системы - индексирование документа -слагается из следующих действий:

1) выявления основного смыслового содержания документа (с учетом точки зрения автора документа и информационных потребностей пользователя системы);

2) описания смыслового содержания документа на информационно-поисковом языке (ИПЯ) и получения соответствующего поискового образа документа (ПОД).

При выполнении ретроспективного поиска производится

- 1) выявление смыслового содержания запроса;
- 2) получение поискового обзора запроса (ПОЗ) на информационно-поисковом языке системы.

Результатами индексирования документов и запросов являются их поисковые образы (ПОД и ПОЗ). Поиск документов по запросу означает сопоставление ПОД и ПОЗ. Качество поиска зависит от критериев смыслового соответствия документа запросу (критериев выдачи).

Различают

- теоретико-множественные критерии, основанные на оценке степени совпадения лексических единиц ИПЯ (слов), используемых в ПОД и в ПОЗ;
- критерии, учитывающие с помощью «весовых коэффициентов» относительную информа-

ционную значимость отдельных лексических единиц, входящих в ПОЗ;

- логические критерии, основанные на использовании логических операций (дизъюнкции, конъюнкции, импликации и др.).

Информационно-поисковые языки (ИПЯ), которые используются в настоящее время, можно разделить на три большие группы:

- классификационные языки;
- дескрипторные;
- комбинированные.

Языки классификационного типа, в свою очередь, делятся по структуре:

- ИПЯ иерархической структуры;
- ИПЯ фасетной структуры;
- эмпирические (неиерархические) языки.

Классификационные системы. В *иерархических* классификационных системах лексические единицы (термины) находятся между собой в отношениях включения. При записи они располагаются в порядке постепенного перехода от общих к более частным. Существуют иерархические системы, в которых рубрики включают по 20 и более подклассов, рубрик и подрубрик в нисходящем порядке. Примером такой иерархической классификационной системы является универсальная десятичная классификация (УДК), широко используемая в библиотечном деле и документальных поисковых системах. УДК охватывает весь спектр знаний.

Шифры УДК, которые можно увидеть на обороте титульного листа всех книг, перед заголовками статей в журналах и сборниках, имеют более чем столетнюю историю. В 1905 г. в Брюсселе на французском языке вышло первое сводное издание таблиц десятичной классификации. Эти таблицы были созданы на основе таблиц десятичной классификации Мельвиля Дьюи, впервые изданных в 1876 г.

Каждый класс (первая ступень деления) в УДК содержит группу более или менее близких наук, например, класс 5 - математику и естественные науки, класс 6 - прикладные науки: технику, в том числе информатику, сельское хозяйство, медицину. Каждая последующая присоединяемая цифра не меняет значения предыдущих, а лишь уточняет их, обозначая более частное понятие. УДК настоящего текста: 681.3:62-52.

В основе *фасетной* классификации лежит многоаспектное распределение понятий какой-либо отрасли науки или техники по однородным взаимно исключающим друг друга фасетам.

Примером *эмпирической* (неиерархической) классификации может быть алфавитно-предметная классификация. Словарный запас такой классификации состоит из упорядоченного по алфавиту множества слов, словосочетаний и фраз естественного языка, обозначающих предметы какой-либо отрасли науки или практической деятельности. Каждому предмету или вопросу при этом отводится только один индекс, собирающий всю информацию относительно данного предмета или вопроса независимо от аспекта рассмотрения. В предметных классификациях используются следующие термины:

- предметная рубрика (заголовок) - слово или фраза естественного языка, используемая для обозначения основного предмета документа (или запроса);
- предметный подзаголовок - слово или фраза, обозначающая аспект рассмотрения предмета, указанного в предметном заголовке или в подклассе предметов, входящих в класс, обозначенный предметным заголовком;
- предметный словник - упорядоченное по алфавиту множество предметных заголовков, используемых для построения какого-либо каталога или указателя.

Система предметных заголовков и подзаголовков и более мелких разделов создает сложную предметную классификацию по аспектам рассмотрения предмета, т.е. имеет некоторые черты фасетных классификаций.

Дескрипторные информационно-поисковые языки. Дескрипторные информационно-поисковые языки основаны на методе координатного индексирования, сущность которого сводится к тому, что смысловое содержание документа может быть с достаточной точностью и полнотой выражено списком ключевых слов, содержащихся в тексте.

Ключевое слово - это лексическая единица информационно-поискового языка, являющаяся существительным, прилагательным, глаголом, наречием, числительным или местоимением естественного языка или словосочетанием. Основным критерий отбора ключевых слов из текста - сте-

пень их полезности для индексирования документа или запроса.

Координатное индексирование выполняется с помощью ключевых слов и логических операции конъюнкции и дизъюнкции. Близкие по смыслу ключевые слова образуют классы условной эквивалентности, имена которых также являются единицами поискового языка и называются дескрипторами. Дескрипторы вместе с набором ключевых слов языка и семантических связей между ними образуют тезаурус - систематизированный набор данных об области знания, позволяющий в ней ориентироваться.

В дескрипторной статье тезауруса обычно устанавливаются следующие виды семантических отношений:

- отношение синонимии;
- отношение подчинения;
- отношение ассоциации.

Такие связи служат для увеличения полноты представления содержания документов и позволяют формировать запрос не обязательно в ключевых словах документа.

Пример дескрипторной статьи:

языки алгоритмические синонимы: алгоритмические языки

машинно-ориентированные языки проблемно-ориентированные языки вышестоящие: программное обеспечение

языки формальные нижестоящие: автокоды

алгол паскаль фортран си бейсик ассоциации: алгоритмы

программирование

Целесообразность применения того или иного языка во многом зависит от назначения информационной системы, степени ее автоматизации. Для описания документов в библиотеках, общих и технических архивах обычно применяют классификационные языки. В автоматизированных информационно-поисковых системах используются, главным образом, дескрипторные языки.

1.3. БАНК ПЕДАГОГИЧЕСКОЙ ИНФОРМАЦИИ

В качестве примера реально существующего и достаточно широко используемого банка данных рассмотрим банк педагогической информации (БПИ), созданный в Республиканском институте повышения квалификации работников образования под руководством В.И.Журавлева. На основе этого банка в ряде регионов России созданы и успешно функционируют региональные банки педагогической информации.

БПИ предназначен для хранения всех видов информации, циркулирующей в системе образования страны, и удовлетворения информационных потребностей пользователей.

С банком работают лица следующих категорий:

- работники образования, получающие с разными целями информацию из банка (к ним относятся учителя, методисты, работники управленческих структур всех уровней, ученые педагоги, преподаватели вузов, студенты и учащиеся, родители и др.);

- поставщики информации, т.е. авторы учебной, учебно-методической и научно-методической литературы, разработчики нормативно-правовой документации, относящейся к системе образования и т.д.;

- работники информационной системы, функция которых состоит в накоплении информационного фонда вторичных документов - информационных модулей, разработанных в соответствии с концепцией БПИ (информационные (информационно-педагогические) модули готовятся на основе первичных материалов, представленных поставщиками информации; такая работа требует специальной подготовки);

- непосредственные работники информационной службы, ведущие лингвистическую часть информационной системы: администратор банка, эксперты.

В перспективе в нашей стране должна быть создана единая сеть банков педагогической информации, объединяющая как центральный республиканский банк, так и региональные БПИ. В последних наряду с информацией общего назначения сосредоточивается информация регионального характера, отражающая опыт учителей и методистов региона, деятельность региональных учреждений повышения квалификации и местных вузов, связанную с региональной системой образования, региональных органов управления образованием и другую информацию педагогиче-

ской направленности. Принципиально важной частью такой сети является такая система обмена данными между различными банками, при которой пользователь любого из них может получать информацию из любого банка и отправлять ее туда.

Содержание и структура банка. Охарактеризуем содержание БПИ приведя перечень и краткое описание нескольких его разделов.

Учебная литература, авторские курсы. Вариативность отечественного образования - характерное его свойство, возникшее в последнее десятилетие. Почти по всем дисциплинам есть несколько вариантов изложения, подкрепленных учебниками, сборниками упражнений, тестов и т.д. Учитель не может иметь все эти материалы, да в этом и нет необходимости, так как он обычно работает по одному из вариантов. Однако, время от времени возникает потребность ознакомиться и с другими вариантами, что можно сделать с помощью банка.

Особенно эта проблема актуальна при необходимости ознакомиться с разработками по специальным (авторским) курсам, к которым зачастую практически нет другого доступа.

Методы обучения. В конце XX века в образовании во всем мире активизировались поиски новых форм и методов обучения. Это связано со стремительными переменами в общественном устройстве (и, в частности, с информационной революцией). Ежегодно публикуется множество материалов на эту тему, аккумулируемых в БПИ.

Аспекты педагогической науки. Новые разделы теории воспитания и описание практических приемов реализации педагогических концепций - важная часть педагогической информации.

Диагностика педагогического профессионализма. Проблемы диагностики успехов обучения были актуальны всегда, но стали еще более актуальными в связи с внедрением государственных образовательных стандартов. Все чаще используются тестовые методики контроля, сравнительно новые для нашей школы; на эту тему постоянно возникают новые материалы.

Зарубежная педагогическая информация. Без знания происходящего в этой сфере в мире невозможно правильно ориентироваться в проблемах образования собственной страны. Прямой доступ к зарубежным материалам для большей части работников отечественной системы образования затруднен по финансовым, языковым, терминологическим и прочим причинам. Аккумулирование в банке обзорных (и оригинальных) материалов позволяет существенно продвинуться в решении этой проблемы.

Инновации в образовании. Многочисленные находки учителей, методистов, работников органов управления часто остаются невостребованными просто из-за невозможности ознакомления с ними широкой педагогической общественности. Даже по отношению к широко известным педагогам-новаторам существуют проблемы доступа к конкретным материалам.

Историко-педагогическое наследие, как отечественное, так и зарубежное, весьма велико, и соответствующие материалы необходимо иметь в БПИ.

Повышение квалификации работников образования - необходимое условие функционирования системы образования. Наряду с давно действующими в этой сфере институтами повышения квалификации, указанной деятельностью занимаются вузы, коммерческие организации и др. Информация на эту тему в БПИ необходима и поможет сделать правильный выбор.

Опыт управленческой работы для системы образования не менее важен чем чисто педагогический опыт. Приемы организации управления учебными заведениями, учебно-воспитательным процессом систематизируются в БПИ.

Правовая основа педагогической деятельности важна для каждого участника педагогического процесса - учителя, ученика, родителей, администраторов. В системе образования существует большое количество нормативно-правовых актов, накапливаемых в БПИ.

В БПИ содержатся также сведения о рынке образовательных услуг, неформальных объединениях молодежи и других аспектах деятельности системы образования.

Как и во всяком «электронном» банке информации, является важным вопрос о структурных единицах хранения информации. Основной такой единицей является информационно-педагогический модуль (ИПМ). ИПМ - формализованное отражение информации, обладающее следующими свойствами:

- логической законченностью (изложенная в одном ИПМ информация соответствует одному и только одному типу);
- информативностью (объем и качество информации в ИПМ достаточны для ее использования в науке или практике без обращения к источнику);

- популярностью изложения (язык изложения доступен педагогу-практику);
- краткостью изложения (объем ИПМ не превышает 5 кбайт).

ИПМ записываются в специальных форматах. Структура и форма записи в них выбрана таким образом, чтобы информацию мог формализовать не только специально подготовленный человек, но и сам автор, что оправдано экономически и способствует формированию информационной культуры работников образования. Поскольку оформление содержательной (т.е. основной) части информации требует знаний в достаточно узких предметных областях образования, никто лучше автора этого сделать не может; администратор же банка (или методист) могут при необходимости оказать автору помощь в некоторых формальных моментах. В то же время, создание ИПМ по источникам, авторы которых не доступны для прямого общения (например, когда ИПМ создается по статьям в зарубежных журналах), могут осуществлять специалисты банка - референты.

Отметим, что подобная практика существует и в научных реферативных журналах, куда может быть помещен либо реферат статьи, написанный референтом, либо автореферат, написанный автором статьи.

ИПМ представляет собой текстовый файл, состоящий из двух полей: поля поискового образа документа и поля записи данных. Каждое из полей распадается на подполя - элементы данных. Рассмотрим структуру и содержание записи отдельно для каждого поля.

Поле поискового образа документа. В этом поле записываются элементы данных, необходимые для поиска информации, ее классификации, анализа наличия или отсутствия, экспертизы по различным параметрам, осуществления коммуникаций как внутри системы, так и с другими автоматизированными информационными системами. Поскольку параметры информации в таких системах жестко стандартизированы, а эти стандарты (ГОСТы) недостаточно полно отражают специфику педагогических исследований, то допускается описание элементов данных, не имеющих аналога в ГОСТах; они помечаются символом «звездочка» (*). Элементы данных, запись в которых (или ее часть) может повторяться в других подполях, помечены символом (1). Максимальная длина записи в элементах данного поля составляет 80 байт (длина строки экрана). В тех случаях, когда элемент данных имеет другое ограничение, оно указано в конце строки в скобках.

Структура записи элементов данных:

- 1) название рубрикатора (*);
- 2) указатель десятичной классификации (УДК);
- 3) название тезауруса (*);
- 4) информационное поле (*);
- 5) фасет (*);
- 6) дескриптор тезауруса (1);
- 7) дата ввода записи;
- 8) организация - создатель записи;
- 9) автор, авторский коллектив (*) (1);
- 10) основное заглавие (250);
- 11) место работы автора (страна);
- 12) место работы автора (область);
- 13) составитель текста записи (1);
- 14) редактор текста записи (1);
- 15) переводчик (1);
- 16) тип информации (*);
- 17) ключевые слова (через запятую) (1);
- 18) вид исходного документа;
- 19) дополнительные данные об исходном документе;
- 20) библиографическая ссылка (250)(*)(1).

Для формирования пунктов 1-3 необходимо пользоваться справочниками по УДК, пунктов 18 и 19 - таблицами кодов «вид документа» и «код дополнительных данных», имеющимися в документации к банку.

Поясним использованное выше понятие «фасет». Поскольку информационное поле охватывает очень широкий объем понятий, то используется фасетный метод классификации: термины

внутри поля располагаются по фасетам - небольшим параллельным группам. Внутри фасеты - дескрипторы, т.е. еще более мелкое дробление. Все это делается для организации эффективного и удобного для пользователя и для системы поиска информации - по классу, по автору, по региону, по ключевым словам. Детальное описание фасетированного рубрикатора находится в документации банка и частично заведено в оболочку пользователя.

Поле записи данных. В этом поле записывается содержательная часть информации, заносимой в ИПМ. В случае, если информация распадается на ряд единичных типов информации (т.е. образует информационный комплекс), поле записи данных распадается на отдельные подполя, в каждом из которых записывается отдельный ИПМ со своим названием. Записи ИПМ в этом случае должны образовывать логическую последовательность в дедуктивном порядке. Например, уместно рассматривать данный учебник как совокупность глав по относительно самостоятельным разделам информатики и при составлении соответствующего ИПМ в общее поле записи данных поместить ИПМ по каждой из глав.

Пример информационно-педагогического модуля. В качестве прообраза, для которого создается ИПМ, рассматривается данный учебник.

Поле поискового образа.

- 1 Народное образование. Информатика.
- 2 681.3
- 3 Тезаурус ЮНЕСКО-МБП по образованию
- 4 Обучение
- 5 Обучение
- 6 Предметное обучение
- 708.02.1999
- 8 Авторский коллектив
- 9 Авторский коллектив в составе: А.В.Могилев, Н.И.Пак, Е.К.Хеннер
- 10 Информатика
- 11 Российская Федерация
- 12 Воронежская область, Красноярский край, Пермская область
- 13 Е.К.Хеннер
- 14 И. П. Сидоров
- 15
- 16 Учебник
- 17 Информатика, теоретические основы информатики, программное обеспечение, языки программирования, архитектура ЭВМ, компьютерные сети, информационные системы, компьютерное моделирование
- 18 Рукопись
- 19 Учебное пособие для подготовки бакалавров по профилю «Информатика» и учителей информатики
- 20 Информатика. / Под ред. Е.К.Хеннера. - М.: Издательский Центр «Академия», 1999

Поле записи данных

«Учебное пособие «Информатика» предназначено для подготовки бакалавров по направлению 540106 «Естествознание», профилю «Информатика», магистров по программе «Информатика в образовании» и учителей информатики (специальность 030100 «Информатика»), а также подготовки по педагогическим специальностям, для которых предусмотрено получение дополнительной квалификации «Учитель информатики». Пособие соответствует государственным стандартам по той части предметных блоков, которая касается информатики и вычислительной техники. В пособие включены материалы по следующим разделам: «Теоретические основы информатики», «Программное обеспечение ЭВМ», «Языки и методы программирования», «Вычислительная техника», «Компьютерные сети и телекоммуникации», «Информационные системы», «Компьютерное моделирование». Материал, включенный в пособие, может обеспечить пяти-шести семестровый курс информатики, изучаемый в педагогических вузах при профильной подготовке по информатике бакалавров, магистров и специалистов -учителей информатики. Пособие может также быть полезным для студентов различных вузов, обучающихся по специальностям, связанным с информатикой. Кроме того, пособие можно использовать при повышении квалификации и переподготовке

учителей».

Поиск информации в банке. Здесь речь пойдет лишь о внешней стороне поиска, т.е. его пользовательском интерфейсе. Скрытые от пользователя механизмы поиска обсуждались в гл. 3 (раздел «сортировка и поиск»).

Работа пользователя с банком начинается с меню, изображенном в верхней строчке рис. 6.1.

Просмтр	Доввод	Справочники	Ехррт	Пароль	Чистка	Выход
По атрибутам						
По дополн. реквизитам						
По каталогам						

Рис. 6.1. Вид меню БПИ в режиме «Просмотр»

В режиме «Пользователь» доступны только два поля меню: «Просмотр» и «Выход». Выбрав пункт меню «Просмотр», получаем подпункты, изображенные на рисунке 6.1. Таким образом, любой материал, имеющийся в банке, можно найти тремя различными путями.

Путь «По атрибутам» открывает наиболее широкие возможности поиска информации, отраженные на рисунке 6.2.

Тезаурус	КлассИнф	Автор	Регион	КлючСлово	Пуск
Тезаурус ЮНЕСКО МПБ			Автор текста		
Поле Фасет Дескрипторы			Регион		
Класс информации			Ключевое слово		

Рис. 6.2. Поиск информации по атрибутам

При поиске по Тезаурусу имеется семь полей, на которые поделено все образовательное пространство:

- вещи (педагогические средства);
- контекст (общество и образование);
- люди (учащиеся и учителя);
- обучение;
- развитие и учение;
- содержание образования и воспитания;
- управление и исследования.

Поиск по фасетам и дескрипторам может происходить: по классу информации, по автору, по региону, по ключевым словам.

Аналогично происходит поиск информации и «По дополнительным реквизитам». При выборе такого пути возникает меню, в котором можно выбирать пункты «Организация», «Составитель», «Редактор», «Дополнительные сведения», «Дата» и продолжать поиск.

Третий путь организации поиска - «По каталогам». Ступив на него, получаем приведенное на рис. 6.3 меню. Выбрав нужный каталог, можем попасть в подкаталоги и т.д., пока не найдем нужного ИПМ или не убедимся в его отсутствии.

Выбор информации по каталогу и подкаталогу
Каталоги
Авторские курсы и другие разработки Активные методы обучения Аспекты педагогической науки Диагностика педагогического профессионализма Зарубежная педагогическая информация Из историко-педагогического наследия Инновации в образовании Неформальные объединения молодежи Опыт управленческой работы Педагогический опыт Повышение квалификации работников образования Правовая основа педагогической деятельности Рынок образовательных услуг

Рис. 6.3. Поиск информации по каталогам

Функции администратора банка. Здесь мы опишем лишь те функции администратора банка, которые обусловлены программной оболочкой. Они отражены в следующем меню:

Просмотр	Доввод	Справочники	Экспорт	Пароль	Чистка	Выход

По существу операции означают следующее:

- просмотр и корректировка информационных материалов;
- дополнительный ввод новой информации;
- работа со справочниками (корректировка, довод);
- создание новой базы с выборкой материалов;
- изменение пароля;
- коррекцию справочников.

После вхождения в пункт меню «Доввод» на экране появляется таблица (рис. 6.4), в которой следует последовательно заполнить позиции; результаты отражаются в колонке «Результаты ввода».

При этом система позволяет воспользоваться рядом встроенных в нее справочников, а также корректировать сами справочники.

Доввод новой информации по позициям заканчивается вводом текста, который должен быть заранее подготовлен в текстовом редакторе (рис. 6.5); при этом необходимо указать диск, на котором находится нужный текст, каталог и название файла.

Данная оболочка позволяет автоматически (раздел меню **export**) создавать новую базу с выбранными материалами или производить обмен текстами между аналогичными базами. Детальное выполнение этих манипуляций можно произвести следуя указаниям последовательно появляющихся меню.

После описанных выше операций необходимо произвести корректировку базы с помощью операции «Чистка». Эта чисто техническая операция описана в документации.

Для предотвращения недозволенного выполнения пользователями функций по управлению банком операции, доступные лишь администратору, защищены паролем, который следует периодически обновлять.

Доввод нового текста	
Дата гг мм дд.	
Поле	
Фасет	
Дескриптор	
Класс информации	
Автор	
Регион	
Тип документа	
Организация	
Составитель	
Редактор	
Дополнит сведения	
Каталог	
Подкаталог	
1 Ключевое слово	
2 Ключевое слово	
3 Ключевое слово	
Заголовок	
1 Помощь 2 3 4 5 6 7 8 9 10 Ввод	

Рис. 6.4. Операции в меню «Доввод»

Доввод			
A B C D	Имя	Размер	Атрибут

Рис. 6.5. Меню для довода текста

Контрольные вопросы

1. Какие виды информационных систем существуют?
2. Какие типы банков информации различают?
3. Каковы основные компоненты документально-поисковой системы?
4. Что называется индексированием документов? Зачем оно производится?
5. Охарактеризуйте основные типы информационно-поисковых языков.
6. Каково предназначение банков педагогической информации?
7. Какие основные разделы области знаний «Образование» отражены в БПИ?
8. Что представляет собой информационно-педагогический модуль и каков его формат?
9. Какие элементарные данные входят в поле поискового образа?
10. Какие способы поиска информации реализованы в БПИ?
11. Каковы функции администратора БПИ?

§ 2. БАЗЫ ДАННЫХ В СТРУКТУРЕ ИНФОРМАЦИОННЫХ СИСТЕМ

2.1. ОСНОВНЫЕ ПОНЯТИЯ

Понятие «база данных» в отличие от понятия «банк информации» изначально связано с компьютерными системами, с историей и развитием.

Исторически понятие базы данных возникло как альтернатива файловой организации данных при хранении с помощью ЭВМ (на магнитных носителях). Такая организация данных была характерна для прикладного программного обеспечения на начальном этапе распространения вычислительной техники. Файловая организация предполагала хранение данных в виде совокупности файлов, ориентированных на использование какой-либо одной прикладной программы, предназначенной для решения некоторой специфической задачи. Такая неуниверсальность в организации информации привела к большой избыточности (дублированию) при хранении, противоречивости данных, хранящихся в различных системах.

Для файловой организации данных свойственна высокая зависимость данных от программ, так что информация о форме хранения данных скрыта в сочетании «файл - программа».

Понятие «база данных» возникло в результате стандартизации и унификации данных, универсально организованных и хранящихся с помощью ЭВМ с целью использования для многих приложений. При этом описание данных уже не скрыто в программах, а явным образом декларируется и хранится в самой базе. База данных может быть определена как структурная совокупность данных, поддерживаемых в актуальном состоянии (в соответствии объектам некоторой предметной области) и служащая для удовлетворения информационных потребностей многих пользователей. Базы данных устраняют избыточность и противоречивость данных. Для поддержания актуальности данных, хранящихся в базе, получения сводок по информационным запросам, перехода к данным и программам пользователей служат системы управления базами данных (СУБД), описанные в общих чертах в гл. 2.

Основой СУБД являются два языка - язык описания данных (ЯОД) и язык манипулирования данными (ЯМД). С помощью языка описания данных администратор базы данных и программисты описывают структуру и содержимое базы данных. Язык манипулирования данными является средством, которое применяется пользователями или прикладным программистом для выполнения операций над данными, хранящимися в базе: добавления новых данных, изменения или удаления устаревших, упорядочения данных по тем или иным признакам, поиска данных в соответствии с запросами.

Интеграция данных в базе подразумевает совместное использование данных для решения различных прикладных задач и устраняет дублирование данных. Однако согласованное понимание и использование данных требует централизованного управления, которое называется администрированием данных. Подчеркнем, что в данном случае идет речь не о собственно значениях данных, а об их смысле и форме.

Смысл элементов данных, способы их использования, источники, различные правила и ограничения представляют собой метаданные. В формализованном виде метаданные могут содержаться в словарях данных, обслуживающих базы данных. Основными функциями словарей данных являются

- установление единообразного понимания данных пользователями БД;
- эффективное управление элементами данных при модификации описания данных в системе;
- уменьшение избыточности;
- устранение противоречивости;
- упрощение проектирования БД и ее сопровождения, расширения и т.д.

Словари данных используются конечными пользователями при работе с системой на языке запросов, прикладными программистами при написании программ, системными программистами в процессе развития системы.

Коллектив специалистов, обслуживающих большие базы данных, включает администратора, аналитиков, системных и прикладных программистов.

Администратор - это специалист, имеющий представление об информационных потребностях конечных пользователей и отвечающий за определение, загрузку, защиту и эффективность базы данных.

Аналитики, обладая знаниями закономерностей соответствующей предметной области, в контакте с конечными пользователями строят формальные (математические) модели для задач конечного пользователя, которые являются исходным представлением задачи для прикладного программиста.

Прикладные программисты на основе представления задачи, полученного аналитиками, разрабатывают прикладные программы для решения задач конечных пользователей.

Системные программисты обеспечивают работоспособность операционной системы, систем программирования и СУБД, разрабатывают сервисные программы.

Приведем перечень важнейших требований, которым должны удовлетворять современные базы данных:

- адекватность базы данных предметной области;
- интегрированность данных;
- независимость данных;
- минимальная избыточность хранимых данных;
- целостность базы данных;

- обеспечение защиты от несанкционированного доступа или случайного уничтожения данных;
- гибкость и адаптивность структуры базы данных;
- динамичность данных и способность к расширению;
- возможность поиска по многим ключам.

2.2. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

Рассмотрим вопрос о проектировании баз данных. К любой базе данных возможен подход на каждом из следующих трех уровней (рис. 6.6):

- на уровне представлений данных конечного пользователя или прикладного программиста (внешнего представления);
- на уровне концептуального представления данных (представления администратора);
- на уровне внутреннего представления данных (с позиции системного программиста) или представления реализации.

Под представлением данных понимаются правила организации и кодирования данных.

Представление данных конечного пользователя (внешнее представление) является совокупностью требований к данным со стороны некоторой конкретной задачи или программы.

Для конечного пользователя внешнее представление является совокупностью спецификаций и реальных форматов данных, отражающих конкретные информационные потребности при решении некоторой задачи (проблемы). Представление прикладного программиста отображает элементы данных и их взаимосвязи так, что из данных, хранящихся в базе, может извлекаться скрытая в них информация (например, возраст человека по дате рождения и текущей дате).

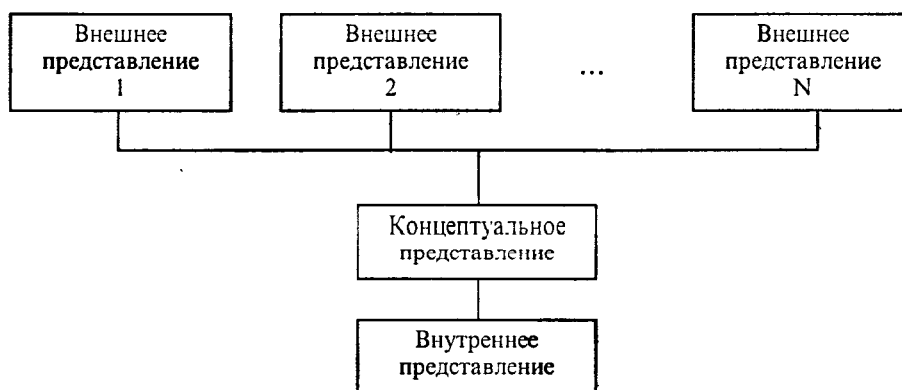


Рис. 6.6. Трехуровневое представление данных в концепции ANSI/SPARC

Так как с данными обычно работают многие пользователи и прикладные программисты, имеется множество частично перекрывающихся внешних представлений данных.

Концептуальное представление данных является интегрированным определением данных на основе объединения внешних представлений данных для всей совокупности приложений, т.е. достаточно полной моделью предметной области. Структура данных на концептуальном уровне называется концептуальной схемой и описывает семантику данных.

Внутреннее (физическое) представление, или представление реализации, выражает представление данных системными программистами и связано с организацией хранения данных на физических носителях информации (запоминающих устройствах) и их обработкой. Основными понятиями внутреннего представления являются физические блоки, хранимые записи, указатели и т.д. Внутреннее представление обеспечивает доступ к данным на логическом уровне и скрывает от прикладных программистов и пользователей многие технические детали манипулирования данными и методов доступа к ним.

Наибольший интерес для нас представляет концептуальное представление данных, связанное с развитой в 70-80-е годы теорией баз и банков данных и направленное на унификацию данных и уменьшение избыточности при интегрировании внешних представлений в концептуальное.

Концептуальное представление основано на определенной модели данных. Этот термин, впервые введенный в 70-х годах основоположником теории баз данных Дж.Коддом, в современ-

ной трактовке отображает совокупность правил порождения структур данных в базах данных, последовательности их изменения. Различают три основные типа модели данных: иерархический, сетевой и реляционный, рассмотренные ранее в гл. 2.

Модель данных предопределяет множество выводимых допустимых типов данных и отношений между ними и является основой для построения модели конкретной базы данных.

Модель базы данных является средством интерпретации содержимого базы данных и реализации операции по обработке и управлению данными.

Проектирование баз данных представляет собой длительный, трудоемкий и слабоформализованный процесс, от которого зависит жизнеспособность и эффективность проектируемой базы данных, ее способность к развитию. Важную роль при проектировании базы данных играет методология построения концептуальных моделей предметной области, включающая методы и средства, позволяющие спроектировать базу данных, удовлетворяющую заданным целям и требованиям пользователей и прикладных программистов.

Таковыми средствами моделирования являются системный анализ, методы экспертных оценок, с помощью которых в концептуальной модели совмещаются концептуальное представление объективно существующей предметной области и концептуальное представление субъективных информационных требований к данным со стороны пользователей и прикладных программистов.

В недавнем прошлом процесс проектирования баз данных был ориентирован, в основном, на требования пользователей и прикладных программистов (ПП-информация) и учитывал текущие или предвидимые приложения. В этом случае база данных создавалась сравнительно легко и быстро. Однако такие базы данных оказывались неприспособленными к обработке неформализованных, изменяющихся, не предвиденных ранее запросов и приложений, не имели стимулов к дальнейшему развитию. Поэтому важную роль при проектировании стала играть информация о предметной области (ПО-информация), не зависящая напрямую от существующих приложений и обеспечивающая гибкость, адаптивность и универсальность данных, пригодность всей системы к развитию и использованию для незапланированных будущих приложений.

Современная методология проектирования баз данных и построения концептуальных моделей основывается на одновременном учете ПО- и ПП-информаций. ПО-информация в этом случае используется для построения первоначальной информационной структуры данных, а ПП-информация - для совершенствования последней с целью повышения эффективности обработки данных.

Процесс построения концептуальной модели разделяется на следующие этапы:

1) сбор и содержательный анализ априорной информации о предметной области и прикладных задачах пользователей;

2) концептуальный анализ данных и синтез концептуальной модели.

На этапе сбора данных проводятся наблюдения и измерения, собираются отчеты и различные документы, интервьюируются специалисты в данной предметной области, выявляется перечень задач организации и ее структурных подразделений. Сбор информации начинается с определения сферы применения базы данных. Сфера применения базы данных должна определяться независимо от прикладных задач и охватывать все функциональные подразделения организации. Для этого проводятся собеседования с руководством организации с целью выявления отделов данной организации и внешних организаций, связанных с информационным обеспечением текущих и планируемых прикладных задач, а также возможных в будущем изменений в деятельности организации. Далее проводятся дополнительные собеседования в подразделениях организации с целью выявления совместно используемых данных. На этом же этапе собранные данные анализируются на предмет устранения дублирования и противоречивости данных, неоднозначности их определений и описаний, выявляются и формулируются правила обработки информации и принятия решений.

Составляется список данных, требуемых для выполнения каждой из производственных или управленческих функций, а также формулируются явные и неявные правила, определяющие, как и когда выполняется каждая функция.

Результатом данного этапа являются

1) список всех создаваемых и используемых элементов данных;

2) перечень прикладных задач, их характеристик и используемых в них данных;

3) список принимаемых решений в управлении организацией или процессами, а также

условий и правил их принятия;

4) список возможных будущих изменений в деятельности и их влияний на принятие решений.

На этапе концептуализации собранной информации выявляются элементы предметной области, их свойства и взаимосвязи, затем синтезируется структура концептуальной модели базы данных. Наиболее известными подходами концептуализации являются анализ сущностей и представление знаний.

Так как собранная на предыдущем этапе информация является плохо структурированной, для ее концептуального анализа необходимо использовать методы системного и классификационного анализа, являющиеся универсальными инструментами организации неформализованного знания.

На первом шаге анализа ПО-информации предметная область разбивается на несколько относительно слабо связанных между собой подобластей. Связи между элементами внутри каждой подобласти являются сильными и реализуют логические отношения типа «род - вид», «целое - часть».

Следующим шагом анализа является декомпозиция каждой подобласти, вычленение компонент (подсистем, частей) и видов (подклассов), связанных такими же отношениями с объектами соседних уровней иерархии.

Декомпозиция должна

- быть направлена на выделение элементов предметной области, существенных с точки зрения прикладных задач базы данных;
- приводить к вычленению элементов, свойства которых могут быть описаны с помощью собранных на первом этапе элементов данных;
- прекращаться при достижении уровня иерархии, исчерпывающего собранную априорную информацию об элементах данных.

Следующим шагом концептуального анализа ПО является анализ существенных свойств и взаимосвязей элементов, выделенных на стадии декомпозиции, а также формулирование и наполнение этих свойств с помощью выявленных ранее элементов данных. При этом собственные свойства элементов рассматриваются как атрибуты, а взаимосвязи между элементами - как k-арные отношения.

Синтез информационной структуры концептуальной модели проводится как композиция (сборка) структуры с учетом связей между частями.

В последние годы мощным инструментом организации плохо структурированных знаний и построения концептуальных моделей сложных предметных областей стал, так называемый, системный анализ. В основе системного анализа лежат принципы системного подхода, являющиеся методологией современного социально-научного познания и социальной практики, составляющие трактовку любого рассматриваемого объекта как системы. Системой называется совокупность элементов, находящихся в существенных отношениях и связях друг с другом. Существенность связей означает, что совместно элементы системы приобретают новое свойство (или функцию), которым не обладает ни один из элементов системы в отдельности. Этим система отличается от сети - совокупности элементов, несущественно связанных между собой, свойства или функции любого элемента сети не зависят от других ее элементов. Появление нового свойства системы, не сводимого к свойствам ее элементов, выражается понятием эмерджентности (целостности).

В качестве основных принципов системного подхода при построении моделей используются

- рассмотрения объекта с различных точек зрения, выявления аспектов изучаемого объекта с учетом их взаимосвязи;
- расчленения объекта на более простые подсистемы (основанием для введения подсистем является то, что связи между подсистемами много слабее, чем между элементами внутри подсистемы, а каждая подсистема много проще, чем вся система в целом);
- выделения иерархических отношений типа «целое - часть» между компонентами системы разных уровней и отношений эквивалентности между компонентами одного уровня.

Системный анализ - это применение системного подхода при обработке конкретной информации и принятию решений. Рассмотренные принципы системного подхода являются и принципами системного анализа.

Их дополняют следующие специфические принципы:

- анализ любого процесса принятия решения должен начинаться с выявления и четкой формулировки целей (желаемых результатов деятельности), которые часто определяются на основе рассмотрения системы более высокого уровня;
- необходимо рассматривать лишь те цели, вероятность достижения которых $p > p_0$ за время $t < t_0$, где p_0 и t_0 - пороги осуществимости цели.

Данные специальные принципы предполагают некую системную стратегию анализа, требующую рассмотрения не только самой системы, но и внешней по отношению к ней среды (надсистемы или метасистемы), и определение границы между ними.

Перспективой развития документальных информационно-поисковых систем и баз данных являются банки знаний, новая концепция информационной системы, использующая результаты исследований и разработок в области искусственного интеллекта.

2.3. ПРЕДСТАВЛЕНИЕ ОБ ЯЗЫКАХ УПРАВЛЕНИЯ РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ ТИПА dBASE

2.3.1. Основные элементы СУБД типа dBASE

Напомним некоторые термины, используемые в теории реляционных баз данных. Будем представлять информацию, составляющую базу данных, в виде совокупности прямоугольных таблиц.

Поле - наименьший поименованный элемент информации, хранящейся в БД и рассматриваемой как единое целое (рис. 6.7).

Поле может быть представлено числом, буквами или их сочетанием (текстом). Например, в телефонном справочнике полями являются фамилия и инициалы, адрес, номер телефона, т.е. три поля, причем все текстовые (номер телефона также рассматривается как некоторый текст).

	Поле 1	Поле 2	...	Поле K-1	Поле K
Запись 1			...		
Запись 2			...		
...	...	...	...	...	...
Запись N			...		

Рис. 6.7. Представление информации в реляционной БД

Запись - совокупность полей, соответствующих одному объекту. Так, абоненту телефонной сети соответствует запись, состоящая из трех полей.

Файл - совокупность связанных по какому-либо признаку записей (т.е. отношение, таблица). Таким образом, в простом случае база данных есть файл.

Все данные в БД разделены по типам. Говоря точнее, вся информация полей, принадлежащих одному столбцу (домену), имеет один и тот же тип. Такой подход позволяет ЭВМ организовать контроль вводимой информации.

При определении полей БД каждое поле может быть отнесено к одному из следующих типов:

- *символьный тип*, обозначение C (character);
- *числовой тип*, обозначение N (numeric);
- *логический тип*, обозначение L (logical);
- *тип даты*, обозначение Data;
- *примечаний*, обозначение Memo.

Каждый тип служит для хранения специфической информации. В поле типа C можно помещать любые имеющиеся на клавиатуре символы: буквы, цифры, знаки препинания и т.д. Символьное поле хранится как текст, выровненный по левому краю. Над символьными полями нельзя выполнять каких-либо вычислений.

В числовом поле могут быть только цифры, знаки «плюс» и «минус», десятичная точка. Числа выровнены по правому краю. Над числовыми полями можно выполнять вычисления.

Поле логического типа содержит величины, которые могут принимать только одно из двух значений: истинно или ложно.

Поле типа даты может содержать реальные даты в представлении, принятом в США (месяц/день/год) или в европейских странах.

Описание всех имен, типов и размеров полей составляют структуру базы данных, которая хранится в памяти вместе с данными. Эту структуру всегда можно просмотреть и при необходимости исправить. Следует различать структуру файла и информацию, содержащуюся в нем. Основное различие между структурой и информацией состоит в том, что структура остается неизменной, а информация меняется при каждом редактировании или вводе.

Семейство СУБД dBASE. Среди СУБД для ПК наиболее распространенным в мире является семейство dBASE фирмы «Ashton-Tate». Такая популярность объясняется отчасти исторически, так как эта СУБД появилась одной из первых на рынке программных продуктов данного профиля; не в меньшей степени этому способствовала удачность заложенных основ конструкции СУБД и командного языка. Одна из наиболее широко используемых (в настоящее время устаревших) версий этого семейства - dBASE III+.

В новой версии dBASE IV концепция СУБД и командный язык получили значительное развитие, позволившее устранить недостатки предыдущей версии, а также существенно обогатить палитру средств обработки данных и улучшить интерфейс пользователя.

К числу СУБД реляционного типа, предназначенных для ПК, относятся семейство dBASE (dBASE II, III, III+ и IV), Clipper, Foxbase, R:BASE, Paradox и др. В этих СУБД записи и, соответственно, поля имеют обычно фиксированную длину (чаще всего длина такой записи достигает 4000...5000 байт). Исключение составляют поля типа Мемо, используемые, например, в СУБД семейства dBASE. Число полей в перечисленных СУБД варьируется от 128 до 1024.

Длина поля зависит от типа поля и может составлять от 255 до 4000 байт для текстовых полей, до 20 байт для числовых полей и имеет фиксированные значения для полей типа даты (8 байт) и логических полей (1 байт). Поле типа Мемо служит для хранения больших массивов текстовой информации и хранится в отдельном файле БД, но воспринимается как поле в составе основного файла БД. Это поле имеет плавающую длину, определяемую объемом введенной текстовой информации, и может достигать 32767 байт.

Большинство СУБД реляционного типа для ПК позволяют создавать файлы с числом записей до 1 млрд. и объемом до 2 Гбайт. Обычно ограничением на число и объем записей, в первую очередь, выступает емкость внешней памяти ПК на магнитных дисках.

В состав многих СУБД, предназначенных для работы на ПК, входят три основных компонента: командный язык, интерпретирующая система или компилятор для преобразования команд к выполняемому виду и средства взаимодействия пользователя с СУБД (интерфейс пользователя).

Командный язык служит для выполнения и обеспечения требуемых операций с данными - позволяет создавать структуры файлов БД и манипулировать данными, создавать прикладные программы, экранные формы ввода и вывода информации и т.д. Структура и возможности языка в значительной степени определяют облик конкретной СУБД, ее возможности. В состав командного языка ПК также обычно входят специальные команды по установке параметров и состояний системы (SET-команды), а также функции, предназначенные для различных видов обработки данных и выполнения вспомогательных действий.

В СУБД команды можно выполнять по одной (после набора с клавиатуры) или группами команд в автоматическом режиме, предварительно записанных в специальный (программный) файл. Команды языка СУБД записываются в текстовой форме, близкой к обычному языку. Для того чтобы ПК могла выполнять такую команду, ее надо преобразовать в вид исполнимых машинных команд.

Существует два принципиально различных способа такого преобразования. В первом способе используется *интерпретирующая система*, которая *преобразует поочередно команды* в исполнимый вид перед их непосредственным выполнением. Во втором способе сначала *вся исходная программа преобразуется* (компилируется) в программу из исполнимых машинных команд и затем эта программа выполняется.

Первый способ выполнения команд СУБД имеет то преимущество, что исходная программа занимает сравнительно немного места и памяти. Кроме того, этот способ позволяет выполнять команды по одной в режиме ввода с клавиатуры или в режиме, управляемом с помощью меню. Однако выполнение исходной программы посредством интерпретации команд происходит достаточно медленно. Второй способ в отличие от первого позволяет выполнять программу значительно

быстрее, но программа, составленная из машинных команд, занимает значительно больше оперативной памяти.

Взаимодействие пользователя с СУБД, в которой используется интерпретирующая система, обычно может осуществляться в режиме, управляемом с помощью меню, и реже, ввода команд с клавиатуры. Первый режим дает возможность пользователям работать с пакетом СУБД, не зная командного языка. Содержание выполняемых команд записывается в позиции меню на естественном языке. Пользователь выбирает нужную позицию меню и нажимает клавишу исполнения команды. Обычно в меню включают достаточно широкий круг команд языка СУБД, позволяющих выполнять многие операции по обработке данных, однако далеко не все. При использовании режима ввода команд с клавиатуры требуется знать их синтаксис и способы применения. Этот режим применяется уже более подготовленными пользователями. Признаком данного режима СУБД служит какой-либо знак (например, точка в левом нижнем углу экрана), который служит приглашением для набора команды. В дальнейшем для простоты будем называть этот режим просто *командным*.

Последние версии таких СУБД, как dBASE, RBASE и др., содержат генераторы прикладных программ, позволяющие пользователям без знания командного языка создавать программы средней степени сложности. Эти средства могут применяться как неподготовленными пользователями, так и профессиональными программистами для уменьшения трудозатрат при разработке прикладных программ.

Как уже отмечалось, первой созданной в этом классе программой была dBASE II, созданная Уэйном Рэтлиффом (США) в 1981г. Варианты этой системы до сих пор используются для целей обучения на учебных ЭВМ. Вскоре эта система была усовершенствована, а ее возможности расширены. Появились СУБД dBASE III и dBASE III+, ставшие на долгое время стандартом систем управления базами данных для персональных компьютеров. Позже получили распространение системы FoxBase, FoxPro и Clipper - все имеющие набор команд, похожий на СУБД семейства dBASE, и использующие ту же организацию данных в файлах с типичным расширением **.dbf** (Data Base File - файл базы данных), СУБД dBASE IV. Языки этих систем послужили основой для создания огромного числа прикладных информационно-поисковых систем, автоматизированных рабочих мест (АРМ) в различных предметных областях.

Система управления базами данных dBASE IV значительно усилена по сравнению с dBASE III+ по многим параметрам (ниже в скобках указаны данные для dBASE III+). В частности, можно создавать двумерные массивы переменных общим объемом до 1170 элементов, может быть открыто до 99 (16) файлов, увеличено максимальное число переменных - 15000 (265), число полей в файлах БД - 255 (128), число открытых индексных файлов -10 (7) и т.д. Как уже отмечалось выше, командный язык dBASE IV значительно расширен и улучшен по сравнению с командным языком dBASE III+.

Включение в язык dBASE IV новых команд позволило существенно улучшить следующие функции пакета СУБД семейства dBASE:

- разработку иерархических систем меню прикладных задач;
- разработку экранных форм ввода и вывода данных;
- организацию ввода и вывода информации в отдельные окна на экране;
- вывод данных на печать;
- возможности по поиску и обработке данных в файлах БД;
- обеспечение работы в среде ЛВС и др.

Команды типичной СУБД для персональных компьютеров предназначены для выполнения следующих функций:

- описания БД;
- наполнения БД;
- редактирования БД;
- поиска информации в БД;
- формирования и вывода на экран или принтер информации в виде отчетов.

Представление о возможностях пакета dBASE IV можно получить при знакомстве с управляющим центром (УЦ), реализующим *интерфейс пользователя*, управляемый меню. В состав УЦ входят шесть меню:

- меню Data («Данные») для создания, изменения структуры файлов БД, ввода, обновления

и отображения данных на экране;

- меню Queries («Запросы») для формирования и выполнения запросов на выборку и обновление данных;
- меню Forms («Формы») для разработки экранных форм ввода и вывода информации;
- меню Reports («Отчеты») для разработки форм отчетов и вывода их на печать;
- меню Labels («Марки») для разработки форм марок и вывода их на печать;
- меню Applications («Прикладные программы») - генератор разработки прикладных программ.

Функции УЦ и расположение меню на экране отражают последовательность работы пользователя при разработке прикладной информационной задачи. Работа обычно начинается с создания структуры файлов БД, которые требуются для ее решения. Возможные типы файлов приведены в табл. 6.1.

После создания структур файлов БД прикладной задачи требуется ввести в них данные.

Ввод и обновление данных в меню «Данные» выполняется на экране в стандартных формах. Эти формы не всегда вполне удобны для пользователя. В УЦ предусмотрено меню «Формы», позволяющее разрабатывать собственные экранные формы для ввода и вывода информации.

Таблица 6.1
Типы файлов и их расширения

Файл	Окно УЦ	Файл разработки	Файл кодов	Исполнимый файл
БД	Data	.dbf	-	-
Запрос на выборку	Queries	.qbe	-	.qbo
Запрос на обновление	»	.upd	-	.upo
Выборка dBASE III+	»	.vue	-	-
Экранная форма	Forms	.sqr	.fmt	.fmo
Отчет	Reports	.frm	.frg	.fro
Марка	Labels	.lbl	.lbg	.lbo
Прикладная программа	Applications	-	.prq	.dbo
SQL-про грамма	»	-	.prs	.dbo
Програма, созданная генератором прикладных программ	»	.app	.prg	.dbo

Отбор информации из одного или нескольких файлов БД можно выполнить с помощью меню «Запросы». Отобранные данные можно выводить на экран в разработанных ранее формах или на печать.

Меню «Запросы» позволяет также связывать между собой несколько файлов БД, что дает возможность при разработке структуры файлов БД прикладной задачи избежать дублирования информации.

Завершающий этап процесса обработки информации - вывод полученных данных на экран, принтер или внешние носители. Для генерации форм выходных документов предназначено меню «Отчеты».

Таким образом, меню УЦ позволяет выполнять все этапы процесса обработки информации, начиная от ее записи на носители и завершая выдачей необходимых выходных документов.

Панель УЦ состоит из шести окон. Окно Data содержит имена файлов БД. Окно Queries показывает имена файлов запросов на выборку и обновление данных. В окно Forms высвечиваются имена файлов экранных форм ввода и вывода информации. Окно Reports содержит имена файлов форм отчетов, а окно Labels - имена файлов форм марок.

Крайнее справа окно Applications высвечивает имена программ или любых текстовых файлов.

Для создания файлов соответствующих типов в окнах панели УЦ имеется позиция <Create> (создать). После выбора этой позиции высвечиваются экраны, предназначенные для выполнения операций по созданию файла соответствующего типа. Эти же экраны раскрываются и при выборе имени файла в окне.

Помимо основных функций работы с файлами различных типов управляющий центр обыч-

но имеет широкий набор сервисных функций, позволяющих работать с каталогами, определять макропоследовательности клавиш, выполнять экспорт и импорт данных в файлы других систем, защищать данные, устанавливать параметры системы, копировать, переименовывать, удалять файлы, запускать DOS-программы и т.д.

2.3.2. Создание структуры файлов базы данных

Рассмотрим теперь, как создать структуру файла БД с помощью УЦ. Для этого надо выбрать позицию <Create> окна Data на панели УЦ. На экране появится таблица, состоящая из шести колонок с именами: порядковый номер поля (Num), имя поля (Field Name), тип поля (Field Type), ширина поля (Width), количество дробных десятичных знаков числа (Dec), признак индексирования поля (Index). Каждая строка этой таблицы содержит информацию об одном поле записи файла БД. Имя поля может включать до 10 буквенно-цифровых символов, но должно начинаться обязательно с буквы. Тип поля (символьный - Character, числовой с фиксированной точкой - Numeric, числовой с плавающей точкой - Float, текстовый - Memo, логический - Logical и типа даты - Date) можно выбрать клавишей ПРОБЕЛ или набрать непосредственно с клавиатуры.

После того как введено описание для всех полей файла БД, созданную структуру следует сохранить на диске. Для этого надо сначала клавишей F10 вызвать меню экрана создания/модификации структуры БД и затем выбрать позицию <Save this database file structure> (сохранить структуру файла БД) меню Layout.

Ввод новых и обновление существующих записей выполняется в одних и тех же стандартных экранных формах Browse и Edit. Предварительно надо файл БД открыть. Факт открытия файла отмечается на панели УЦ тем, что его имя перемещается выше черты под слово <Create>. Находясь на экране разработки структуры файла БД, после ее окончания можно сразу перейти к вводу данных посредством выбора позиции <Enter records from keyboard> меню Append.

Кроме того, ввод данных в стандартные экранные формы можно выполнить из УЦ. Для этого после установки курсора на имя файла в окне Data надо нажать клавишу ENTER и затем выбрать позицию <Display data>. После этого высветится одна из стандартных экранных форм: Browse или Edit.

В форме Browse данные высвечиваются в табличном виде (строка таблицы - запись файла), при этом не все поля могут одновременно разместиться на экране.

В форме Edit поля одной записи размещаются одно под другим на одной или нескольких страницах экрана (в зависимости от количества полей). Раскрытие последующих страниц выполняется клавишей PgDn.

После того как создана структура файла БД и введена информация, обычно требуется отбирать и выводить на экран или на печать те данные, которые отвечают поставленной задаче. Для этого формируются запросы на отбор необходимых данных.

Запросы можно создавать с помощью шаблона файла (запросы по образцу) и в специальном окне, которое размещается в окне Queries панели УЦ.

Формирование шаблона вывода целесообразно выполнять в два этапа: сначала вводить сразу все поля файла и затем удалять ненужные.

Кроме обычных полей в шаблон вывода можно помещать вычисляемые поля.

В пустую графу шаблона вычисляемых полей вводится с клавиатуры вычисляемое выражение, которое включает одно или несколько имен полей файла БД, и операторы выполнения действий с ним. В число этих операторов входят арифметические и логические операторы, а также функции dBASE.

Имеется возможность разрабатывать собственные формы ввода и вывода данных на экран. В этих формах поля файлов БД можно размещать в любом месте экрана, сопровождая их пояснительными надписями, обводить в рамки и вводить другие оформительские элементы.

Разработанные формы представляют собой программные файлы, которые можно записать на диск и затем использовать многократно для просмотра и обновления информации на экране в удобном для пользователя виде.

Разработка экранной формы начинается с выбора позиции <Create> окна Forms панели УЦ. После этого высвечивается горизонтальное меню, включающее позиции <Layout>, <Fields>, <Words>, <Goto>. Выбор этих позиций приводит к открытию различных меню, позволяющих вы-

полнять определенный круг действий по созданию экранной формы.

Так, меню Layout предназначено для общего оформления экранной формы, реализации короткого способа ввода всех полей на экран и назначения файла БД, если он не был открыт ранее.

Добавление полей в экранную форму по одному осуществляется с помощью позиции <Add field> меню Fields.

Разработка отчетов является завершающим этапом технологии обработки информации. При выдаче отчетов могут использоваться выборки данных, полученные в результате выполнения запросов, и исходные файлы БД.

Разработка отчетов выполняется в окне Reports УЦ. Форма отчетов создается на экране. В нее можно включать следующие элементы: поля из текущего файла БД или выборки, вычисляемые поля, которые создаются специально для отчетов, текстовый материал, рамки и линии. Текст длинных символьных полей можно размещать и разворачивать в колонках заданной ширины.

В число стандартных частей отчета входят: заголовок страницы, введение (заголовок) отчета, собственно отчет (тело отчета), итоги (заключение) отчета и подножие страницы.

Описанный УЦ позволяет выполнять многие операции без знания командного языка dBASE IV с помощью режима управляемого меню, однако основной режим работы СУБД основан на использовании командного языка.

2.3.3. Командный язык СУБД

Операции СУБД реализованы в виде команд, которые, как правило, можно использовать независимо, вводя их с помощью клавиатуры - это, так называемый, командный режим СУБД. Из этих команд можно составлять прикладные программы, создавая таким образом системы «под ключ», ориентированные на неподготовленного пользователя. Это, так называемый, программный (пакетный) режим работы в СУБД.

В состав элементов командного языка входят команды, SET-команды, функции, переменные и массивы.

Команда записывается в виде строки слов, в которую входит глагол - основное слово, определяющее содержание операции, выполняемой командой, список выражений и ключевые слова с аргументами. Структура командной строки называется синтаксисом команды. Многие команды обработки файлов строятся по следующей форме:

```
<глагол> [<список выражений:»] [<границы>]  
[FOR<условие>] [WHILE<условие>]  
[TO FILE <имя файла> /TO PRINTER  
          /TO ARRAY <имя массива>  
          /TO <переменные>  
[ALL [LIKE/EXCEPT <шаблон>]] [IN <алиас>]
```

Квадратные скобки означают использование заключенных в них элементов синтаксиса по выбору. В угловых скобках, следующих после ключевого слова, записываются его аргументы. Косая черта, наклоненная влево (/), обозначает, что может быть записано только одно из разделенных этим знаком ключевых слов. Угловые, квадратные скобки и косая черта в записи команды для ее выполнения не указываются и служат только для пояснения синтаксиса.

<Список выражений> представляет собой комбинацию имен полей файлов БД, переменных, элементов массивов, констант, функций, системных переменных и операторов. Все данные в выражении должны быть одного типа. Если в выражении требуется использовать данные разных типов, их надо преобразовать в один тип специальными функциями. Выражения в списке могут быть различного типа.

Операторы в выражениях обозначают операции с данными и обозначаются специальными символами. В dBASE используются четыре типа операторов: математические, операторы отношений, логические и строковые.

Границы в приведенном выше синтаксисе команды указывают те записи файла БД, которые будут доступны при его обработке. В качестве ключевых слов границ могут быть: RECORD <n> -

доступна только одна запись с заданным номером; NEXT <n> - доступны все записи, начиная с номера n; ALL - доступны все записи файла БД; REST - доступны записи, начиная с текущей до конца файла.

Опции FOR и WHILE определяют условия отбора записей для обработки. Эти опции имеют приоритет перед границами при определении доступа к записям. Опция FOR определяет доступ к обработке, начиная с первой записи и до конца файла БД, а WHILE - начиная с текущей записи и до тех пор, пока условие имеет логическое значение «Истина» (True или .T.).

Опция TO управляет выводом информации, обработанной командой. В ряде команд выходной поток можно направить в файл на диске, на принтер, в массив (или переменную).

Опция ALL LIKE позволяет отобрать или, наоборот, исключить среди имен файлов, полей файлов БД и переменных те, которые соответствуют заданному шаблону.

Особой разновидностью команд являются SET-команды, устанавливающие параметры среды, в которой выполняются обычные команды. Применяются две формы этих команд:

SET <параметр> TO <выражение> и
SET <параметр> ON/OFF

Первая форма устанавливает значение параметра, а вторая включает (ON) или выключает (OFF) его действие, т.е. устанавливает состояние. Эти команды можно выполнять в программах, в командном режиме, можно также занести их в специальный файл с именем Config.db, определяющий параметры среды на весь сеанс работы с пакетом dBASE.

Создание файла БД осуществляется командой CREATE <имя файла>. Для модификации созданной структуры файла используется команда MODIFY STRUCTURE.

Для открытия файла БД используется команда

```
USE  [<имя файла БД>/?] [IN <номер рабочей области>]
      [INDEX <список .ndx и .mdx файлов>]
      [ORDER [TAG] <имя .ndx файла>/<указатель .mdx файла>]
      [OF <имя .mdx файла>]]
      [ALIAS<алиас>] [EXCLUSIVE] [NOUPDATE],
```

которая открывает существующий файл БД и связанные с ним файлы .mdx и .ndx в текущей рабочей области.

Для создания индексных файлов типа .ndx и .mdx используется команда

```
INDEX ON <индексное выражение>
      TO <имя .ndx файла>/
      TAG <имя указателя индекса .mdx файла>
      [OF <имя .mdx файла>] [UNIQUE] [DESCENDING]
```

Для открытия индексных файлов используется команда

```
SET INDEX TO [/?<список имен файлов>
      [ORDER [TAG] <имя .ndx файла>/
      <имя указателя индекса>
      [OF <имя .mdx файла>]]]
```

Помимо широкого набора команд типичный язык СУБД типа dBASE персонального компьютера включает несколько десятков или даже сотен встроенных *функций*. Функции могут использоваться на месте переменных и констант в командах. После указания имени функции обязательно идут круглые скобки, даже если функция не требует никаких параметров. В табл. 6.2 приведены некоторые функции, часто используемые при разработке программ.

Таблица 6.2
Некоторые функции СУБД типа dBASE

Функция	Назначение
RECNO()	Возвращает номер текущей записи открытого файла БД
BOF() EOF()	Возвращают логический признак начала и конца открытого файла БД
RECCOUNT()	Возвращает число записей в открытом файле БД
FOUND()	Возвращает логический признак успешности поиска в БД командой SEEK
LIKE(<образец>, <выр.С>)	Сравнивает строку <образец> с символьным выражением <выр.С>
UPPER (<выр.С>)	Преобразует символьную строку в верхний регистр
INKEY()	Возвращает код нажатой клавиши
ERROR()	Возвращает номер ошибки
CHR(<выр.N>)	Возвращает букву по коду
DATE()	Возвращает текущую дату
ROW() COL()	Возвращает номер текущей строки и столбца экрана

В языке dBASE IV используются также функции, которые (наряду с командами) позволяют обрабатывать данные (в том числе, выполнять вычисления), а также определять состояния, например, находится ли указатель записей в конце файла: ?EOF(). После имени функции обычно стоят круглые скобки, в которых могут помещаться аргументы. Функции нельзя использовать самостоятельно, а только вместе с какой-либо командой.

Кроме системных функций в dBASE IV можно использовать функции, созданные самим пользователем.

В языке dBASE используются *переменные*, тип которых автоматически определяется первым в данной программе присвоением им значения. Возможные типы: символьный, логический, числовой, дата.

Кроме простых переменных, в программах и процедурах можно использовать одно- или двумерные *массивы*. Каждый элемент массива представляет собой обычную переменную. Представление переменных в виде массива позволяет значительно расширить диапазон применения переменных, а также, в ряде случаев, облегчить реализацию функций, присущих переменным. Перед использованием массивов сначала их надо объявить командой

```

DECLARE    <имя массива 1>
           [{<количество строк>}, {<количество колонок>}]
           {,<имя массива 2>
           [{<количество строк>}, {<количество колонок>}]}...
```

В синтаксисе этой команды опции указываются фигурными скобками вместо обычно принятых квадратных. При записи команды в программе в квадратные скобки заключаются параметры (размерность) массива - число строк и столбцов. Если указано одно число, то массив одномерный. Формально двумерный массив с одним из параметров, равным единице, фактически является одномерным: DECLARE [12,1] или DECLARE [1,25].

Примеры.

```

DECLARE Mas [8,4]           && Двумерный массив
DECLARE Masi [10]          && Одномерный массив
STORE {15/10/89} TODatI [4,4].]
```

Для ввода данных в переменную или элементы массива в программе используется команда

STORE <выражение> TO <список переменных>/<список элементов массива> или альтернативная форма

<переменная>/<элемент массива> = <выражение>.

2.3.4. Ввод данных в базу и редактирование

После того как создана структура файла БД, можно вводить данные. Существуют две стандартные экранные формы ввода (и вывода) информации в файлы БД: BROWSE и EDIT. Первая форма вызывается командой

```
BROWSE    [NOINIT] [NOFOLLOW] [NOAPPEND] [NOMENU]
           [NOEDIT] [NODELETE] [NOCLEAR] [COMPRESS]
           [FORMAT] [LOCK <Выр.N>] [WIDTH <выр.N>]
           [FREEZE <имя поля>] [WINDOW <имя окна>]
           [FIELDS <имя поля 1> [/R] [/<ширина колонки>]
           /<вычисл. имя поля 1> = <выражение 1>
           [ ,<имя поля 2> [/R] [/<ширина колонки>]
           /<вычисл. имя поля 2> = <выражение 2>]...
```

В команде имеется ряд опций, позволяющих устанавливать различные режимы ее использования. В число этих опций входит группа опций, позволяющих назначить режимы работы с полями, и группа опций по размещению полей на экране. В состав первой группы входят следующие опции: NOEDIT, позволяющая установить режим только чтения для всех полей, NOAPPEND, не разрешающая добавлять новые записи, и NODELETE, не разрешающая помечать записи для удаления. Опция FREEZE назначает для обновления только одно указанное поле. Другие поля высвечиваются, но остаются недоступными для редактирования. Если для указанного в опции FREEZE поля командой PROTECT открыто только чтение, то обновление не разрешается. Опция FREEZE без имени поля отменяет ранее назначенную установку.

Поля высвечиваются в том порядке, который определен структурой файла БД или указан в команде опцией FIELDS, входящей в состав второй группы. Эта опция позволяет выбрать поле и определить порядок его расположения в таблице Browse.

Опция WIDTH устанавливает верхний предел ширины колонки для всех полей в таблице BROWSE. Эта ширина перекрывает ширину поля, назначенную при создании файла БД. Если опции WIDTH и <ширина колонки> используются одновременно, то принимается наименьшее значение. Данная опция, как и предыдущая, не применяется для полей типа Мемо и логических полей. Значения числовых полей и полей даты не будут высвечиваться, если значение опции WIDTH меньше ширины этих полей, назначенных при определении структуры файла БД. Опция FORMAT позволяет разместить поля в таблице BROWSE в очередности, определенной экранной формой, которая установлена командой SET FORMAT TO.

Альтернативой команды BROWSE является команда EDIT

```
EDIT      [NOINIT] [NOFOLLOW] [NOAPPEND] [NOMENU]
           [NOEDIT] [NODELETE] [NOCLEAR] [<номер записи>]
           [FIELDS <спснок полей>] [<диапазон>]
           [FOR <условие>] [WHILE <условие>]
```

Эта команда выводит на экран по одной записи файла БД, поля которой размещаются по вертикали.

Если требуется обновить содержимое полей файла БД без привлечения данных из другого файла, например умножить содержимое числового поля на заданную 'величину, соединить содержимое двух символьных полей и т.д., можно использовать команду

```
REPLACE   [<границы>] <поле> WITH <выражение>
           [, <поле> WITH <выражение...>]
           [WHILE <условие>] [FOR <условие>] [ADDITIVE]
```

которая изменяет (замещает) значения заданных полей (включая поля Мемо) текущей записи или группы записей, определенных границами и опциями WHILE, FOR.

После ввода данных в файл БД нередко требуется быстро их просмотреть. Для этого служат команды

LIST/DISPLAY [*<границы>*] [[FIELDS] *<список выражений>*]
[OFF] [*<диапазон>*] [FOR *<условие>*]
[WHILE *<условие>*] [TO PRINTER/
TO FILE *<имя файла>*]

Эти команды выводят все поля файла БД, если их список не определен опцией FIELDS. В начале каждой записи, если не указана опция OFF, выводится ее номер, опции *<диапазон>*, FOR и WHILE определяют диапазон выводимых записей. Команды LIST и DISPLAY идентичны с той разницей, что первая выводит записи непрерывно, а вторая делает остановку после вывода одной страницы экрана.

Кроме выводов файлов БД команды LIST и DISPLAY позволяют также высвечивать данные о файлах в директориях, содержимое буфера HISTORY (хранилище выполненных с клавиатуры команд), данные об используемых переменных и массивах, информацию о текущем сеансе работы с dBASE, структуру указанного файла БД и имена рабочих станций, работающих в среде ЛВС.

2.3.5. Дополнительные операции

Нередко возникает необходимость *просмотреть структуру файла* и вывести ее на печать. Для этого служит команда

LIST/DISPLAY STRUCTURE [IN *<алиас>*] [TO PRINTER/TO FILE *<имя файла>*]

которая выводит информацию о файле БД.

Существуют специальные команды перемещения по файлу, добавления новых записей и пометки записей для их последующего удаления.

Для *перемещения по файлу* используется команда

GO/GOTO BOTTOM/TOP [IN *<алиас>*]

или

GO/GOTO [RECORD] *<номер записи>* [IN *<алиас>*]

или

<номер записи>

Эти команды осуществляют переход на указанную запись в активном файле БД.

В программах часто требуется *перемещать указатель записи* не на конкретную запись, а на следующую после текущей. Эту операцию выполняет команда

SKIP [*<Выр.N>*] [IN *<алиас>*]

После того как указатель записи перемещен на нужную запись, которая становится текущей, можно выполнить требуемую операцию, например ввести запись до или после текущей. Для этого используется команда

INSERT [BEFORE] [BLANK]

которая позволяет ввести новую запись (или пустую запись, если указана опция BLANK) после текущей или перед ней (если задана опция BEFORE). Пометить одну или несколько записей для их последующего удаления можно командой

DELETE [<диапазон>] [FOR <условие>] [WHILE <условие>]

Опции <диапазон>, FOR и WHILE позволяют *помечать записи*, отобранные в соответствии с заданным диапазоном и условиями.

Для того чтобы *уничтожить пометки* на удаление записей используется команда

RECALL [<диапазон>] [FOR <условие>] [WHILE <условие>]

Физическое удаление помеченных записей активного файла БД выполняет команда PACK.
Команда

APPEND [BLANK]

позволяет *добавлять записи*, введенные с клавиатуры в конец активного файла БД.

В ряде случаев требуется *создать физически упорядоченный файл*. Для этого используется команда

SORT TO <имя файла> ON <поле 1 > [/A] [/C] [/D]
 [,<поле 2> [/A] [/C] [/D]...] [ASCENDING]/[DESCENDING]
 [<границы>] [WHILE <условие>] [FOR <условие>]

которая создает новый файл БД с записями, отсортированными в алфавитном, хронологическом или числовом порядке по заданным полям. Сортировка выполняется в порядке возрастания (/A) или убывания (ID) значений полей. Допускается сортировка максимум по десяти полям. Опция /C определяет сортировку независимо от прописных и строчных букв. Команда

- COPY FILE <имя файла 1> TO <имя файла 2>

делает *копию файла* любого типа.

После копирования или в других случаях нередко требуется *переименовать файл*.

Для этого используется команда

RENAME <имя файла> TO <новое имя файла>

Нередко требуется найти запись не по ее положению в файле БД, а по содержанию полей. Такой вид поиска предусмотрен, например, в меню команды BROWSE. Способ поиска по содержанию полей зависит от того, созданы ли индексные файлы по этим полям или по их комбинациям. Для *поиска по неиндексированным полям* используется команда

LOCATE [FOR] <условие> [<диапазон>] [WHILE <условие>]

Эта команда выполняет в активном файле БД поиск первой записи, удовлетворяющей условию, заданному опцией FOR. Опции <диапазон> и WHILE определяют диапазон и условие начала и конца просмотра записей.

Для *поиска по индексированным полям* используются команды

SEEK и FIND.

Команда

SEEK <выражение>

выполняет поиск первой записи файла БД с заданным значением индексированного поля. При этом соответствующий индекс должен быть установлен в качестве главного (управляющего) индекса. Команда

FIND <литеральная строка>

аналогична команде SEEK и также выполняет *поиск первой записи в индексированном файле БД с заданным значением индекса* при установленном соответствующем главном индексе. Отличие состоит в том, что в качестве поискового значения в команде FIND используется не выражение, а литеральная строка.

Основной командой, предназначенной для *создания нестандартных экранных форм ввода и вывода информации*, является

```
@<строка>,<колонка>
[SAY <выр.>
  [PICTURE <выр.С>]
  [FUNCTION <список функций>]]
[GET <переменная>
  [[OPEN] WINDOW <имя окна>] [PICTURE <выр.С>]
  [FUNCTION <список функций>]
  [RANGE [<нижний>] [<верхний>]]
  [VALID <условие> [ERROR <выр.С>]]
  [WHEN <условие>] [DEFAULT <выражение>]
  [MESSAGE <выр.С>]]
  [COLOR [<стандартная область экрана>] [<нестандартная область
  экрана>]]
```

Эта команда позволяет выводить или вводить информацию в заданном формате в указанные места экрана. Кроме того, с помощью опций SAY этой команды можно выводить данные на принтер или в файл на диске. Для того чтобы установить вывод информации с помощью данной команды на экран, принтер или в текстовый файл DOS, используется команда

SET DEVICE TO SCREEN/prINTER/file <имя файла>

Опция GET выводит и позволяет *обновлять и редактировать данные*, содержащиеся в полях, переменных или массивах.

Собственно *обновление данных* в переменных GET (здесь и ниже под словом «переменная» подразумеваются также поля файла БД и элементы массива) выполняется командой READ, которую следует поместить после данной команды. Эта команда активизирует переменные и выводит в них значения, находящиеся на экране.

Для *приостановления процесса последовательного выполнения программы* используется команда

WAIT [<сообщение>] [TO <переменная>]

которая вызывает паузу до нажатия любой клавиши и высвечивает в нижней строке экрана сообщение. После нажатия клавиши выполнение программы продолжается, а в переменную заносится код ASCII этой клавиши.

С целью *улучшения восприятия информации* на экране можно использовать команду

```
@<строка 1>,<колонка 1> TO <строка 2>,<колонка 2>
[DOUBLE/PANEL/<символы определения рамки>]
[COLOR <код цвета>]
```

которая рисует на экране прямоугольную рамку, определяемую координатами верхнего левого - <строка 1>,<колонка 1> - и правого нижнего - <строка 2>,<колонка 2> - углов рамки. Опция DOUBLE позволяет рисовать двойную линию рамки (по умолчанию рисуется одинарная линия), опция PANEL - рамку в виде сплошной полосы. Кроме того, имеется возможность рисовать ли-

нию рамки заданными символами. Для этого задается список символов для определения рамки. Можно задать различные символы для сторон и углов рамки. В этом случае символы разделяются запятыми и располагаются в следующем порядке: f, b, l, r, fl, fr, bl, br, где f - верх; b - низ; l - левая сторона; r - правая сторона; fl - левый верхний угол; fr - правый верхний угол; bl - левый нижний угол; br - правый нижний угол.

Пример.

@ 5,13 TO 10,40 DOUBLE COLOR R/BG Д

Для *полной очистки всего экрана* используется команда

CLEAR [ALL/FIELDS/GETS/MEMORY/MENUS/
POPUPS/TYPEAHEAD/WINDOWS]

Она очищает экран, устанавливает курсор в левый нижний угол экрана и освобождает все незаконченные опции GET команды @... SAY. ..GET. Различные опции этой команды также закрывают файлы БД, очищают переменные, списки полей, окна, горизонтальные и вертикальные меню и буфер клавиатуры.

Команда @... SAY. ..GET является универсальной для *организации ввода и вывода информации* на экране. В программах нередко требуется вводить с клавиатуры данные для обеспечения процесса обработки информации, например имя файла БД, данные для ветвления программы, поисковые значения и т.д. Для этого предусмотрена специальная команда

ACCEPT [<сообщение>] TO <переменная>

которая позволяет вводить с клавиатуры данные символьного типа в переменную. Перед полем переменной можно высветить сообщение, например, о характере вводимых данных.

Кроме этой команды для *ввода данных с клавиатуры* можно использовать команду

INPUT [<сообщение>] TO <переменная>

которая выполняет аналогичные функции, но позволяет вводить в переменную с клавиатуры данные не только символьного, но и других типов, а также в виде заданного выражения, что не допускается в команде ACCEPT. Тип выражения определяет тип переменной, в которую вводится значение. После набора выражения с клавиатуры для ввода его в переменную надо нажать клавишу Enter.

Кроме универсальной команды SAY существует команда, специально предназначенная для *вывода данных на экран и принтер*:

??? [<выр.1> [PICTURE <выр.С>]
 [FUNCTION <список функций>] [AT <выр.Н>]
 [STYLE <номер шрифта>]] [,<выр.2>...] [,]

которая выводит значение одного или более заданных символьных выражений. Если установлено состояние SET PRINT ON, то выполняется вывод на принтер, если SET PRINT OFF, то на экран. Эта команда в форме ? выводит каждое выражение на следующей строке, а в форме ?? все выражения на одной строке.

Для *печати отчета* используется команда

REPORT FORM <имя файла отчета>/?
 [<границы>] [WHILE <условие>]
 [FOR <условие>] [PLAIN]
 [HEADING <выр С>] [NOEJECT]
 [TO PRINTER/TO FILE <имя файла>] [SUMMARY]

которая запускает программу печати отчета (файл отчета), созданную командой CREATE/MODIFY REPORT.

Для создания исходных программных файлов используются обычные текстовые редакторы, например встроенный редактор dBASE, который вызывается командой

MODIFY COMMAND/FILE <имя файла> [WINDOW <имя окна>]

Прикладная программа обычно состоит из главной процедуры и вызываемых из нее процедур, например:

<команды>	&& Начало главной процедуры
DO Proc1	
DO Proc2	
DO Proc3	
RETURN	&& Конец главной процедуры
PROCEDURE Proc1	&& Начало Proc1
<команды>	
RETURN	&& Конец Proc1
PROCEDURE Proc2	&& Начало Proc2
<команды>	
RETURN	&& Конец Proc2
PROCEDURE Proc3	&& Начало Proc3
<команды>	
RETURN	&& Конец Proc3

Процедура представляет собой программный модуль, который выполняет определенную задачу. После ее выполнения управление передается в вызвавшую или главную процедуру. Процедура так же, как и программа, вызывается и выполняется командой

DO <имя программы>/<имя процедуры> [WITH <список параметров>]

Опция WITH передает в вызываемую процедуру параметры (до 64).

Каждая процедура, размещаемая в файле процедур или в программном файле, должна начинаться с команды

PROCEDURE <имя процедуры>

и завершаться командой

RETURN [<выражение>/TO MASTER/TO <имя процедуры>]

Команда RETURN удаляет все переменные, которые были определены в процедуре как PRIVATE (переменные, определенные как PUBLIC, остаются без изменения).

Командой DO в процедуру можно передавать из вызывающей программы и обратно параметры, которые в самой процедуре определяются командой

PARAMETERS <список параметров>

которая должна быть первой исполняемой командой в теле процедуры. В качестве параметра в списке может быть любое допустимое выражение, определяющее переменную, в которую передаются данные из соответствующего параметра вызывающей программы. Если эта переменная не объявлена глобальной (PUBLIC), то автоматически она имеет статус локальной (PRIVATE).

Определение переменных как локальных выполняется командой

PRIVATE <список переменных> / [ALL [LIKE/EXCEPT <шаблон>]]

Глобальные переменные можно использовать повсюду а программе и в вызываемых ею процедурах. Для объявления переменных и элементов массива глобальными используется команда

```
PUBLIC <список переменных>/ [ARRAY <список элементов массива>]
```

2.3.6. Организация системы меню

Чтобы организовать в dBASE систему меню, сначала надо определить его отдельные компоненты - меню вертикального и горизонтального типа. Для этого используются специальные команды, начинающиеся словом DEFINE - определить. Определение горизонтального меню выполняется командой

```
DEFINE MENU <имя меню> [MESSAGE <выр.С>]
```

Эта команда еще не создает горизонтальное меню, а только назначает ему имя и, если требуется, сообщение, которое будет высвечиваться в нижней строке экрана (строке сообщений). Для определения позиций горизонтального меню используется команда

```
DEFINE PAD <имя позиции>  
  OF <имя меню> PROMPT <выр.С>  
  [AT <строка>,<колонка>] [MESSAGE <выр.С>]
```

которая определяет одну позицию горизонтального меню. Для задания нескольких позиций данная команда повторяется соответствующее число раз. Вертикальное меню определяется командой

```
DEFINE POPUP <имя меню>  
  FROM <строка 1>,<колонка 1>  
  [TO <строка 2>,<колонка 2>]  
  [PROMPT FIELD <имя поля>]  
  /PROMPT FILES [LIKE <шаблон>]  
  /PROMPT STRUCTURE]  
  [MESSAGE <выр.С>]
```

которая устанавливает его имя (записывается по тем же правилам, что и для горизонтального меню) и координаты окна меню на экране. Опции FROM и TO определяют координаты левого верхнего и правого нижнего углов окна меню.

Назначение текста и других атрибутов одной позиции ранее определенного вертикального меню осуществляется командой

```
DEFINE BAR <номер строки> OF <имя меню>  
  PROMPT <выр.С>  
  [MESSAGE <выр.С>]  
  [SKIP [FOR <условие>]]
```

Чтобы определить нескольких позиций, надо эту команду повторить соответствующее число раз.

После того как определены позиции меню, обычно требуется назначить им определенные действия, которые будут выполняться при их выборе. Эти действия для позиций горизонтального меню назначаются командой

```
ON SELECTION PAD <имя позиции горизонтального меню>  
  OF <имя горизонтального меню> [<команда>]
```

которая выполняет заданную команду при выборе указанной позиции горизонтального меню. В качестве команды может быть любая команда dBASE IV, в том числе команда вызова программ и процедур.

Для назначения действий позициям вертикального меню используется команда

```
ON SELECTION POPUP <имя позиции вертикального меню>/ALL [<команда>]
```

которая выполняет заданную команду или процедуру при выборе позиции вертикального меню.

Обычно для назначения позициям меню каких-либо действий используется выход на процедуру, в которой помещаются команды для выполнения этих действий. В такой процедуре чаще всего применяется команда

```
DO CASE
CASE <условие>
<команды>
[CASE <условие>]
<команды>
[OTHERWISE]
<команды>
ENDCASE
```

Можно формировать вложенные структуры этой команды, т.е. внутри команды после ключевого слова CASE помещать другую команду DO CASE, а также любые другие команды ветвления и выполнения циклов, например IF...ENDIF, DO WHILE...ENDDO и др.

Пример.

```
DO CASE
CASEVal> 1000
<команды>
CASEVal> 100
<команды>
CASEVal>10
<команды>
OTHERWISE
<команды>
ENDCASE
```

Если требуется разветвить процесс только на два направления, можно использовать команду

```
IF <условие> <команды> [ELSE] <команды> ENDIF
```

При реализации операций, назначаемых позициям меню, нередко требуется организовать циклы повторяющихся действий, например организовать обработку файла БД и др. Это можно сделать с помощью ряда команд и их комбинаций. Одной из таких является команда

```
DO WHILE <условие>
<команды>
[EXIT]
<команды>
[LOOP]
<команды>
ENDDO
```

которая обеспечивает повторение последовательности команд, заключенной между слова-

ми DO WHILE и ENDDO до тех пор, пока заданное условие истинно. Если встречается слово LOOP, то управление передается в начало цикла команд, а при встрече слова EXIT осуществляется выход из команды и управление передается на первую команду после слова ENDDO, завершающего данную команду.

Предыдущие команды позволяют определить меню и атрибуты их позиций. Для того чтобы активизировать горизонтальное и вертикальное меню (а также списки), используются команды

ACTIVATE MENU <имя меню> [PAD <имя позиции>]

и

ACTIVATE POPUP <имя меню>

которые вызывают на экран предварительно определенные меню.

2.3.7. Пример создания информационной системы с помощью СУБД типа dBASE

Рассмотрим пример создания информационной системы обработки школьной информации «Всеобуч» с помощью СУБД типа dBASE. Проследим основные этапы работы.

1. Постановочная часть.

1.1. Цель функционирования ИС.

1.2. Структурирование общей задачи.

1.3. Анализ входной информации.

1.4. Анализ переработки информации.

1.5. Анализ выходной информации (структура БД, формат экрана для выходных данных, выходные формы информации, стандартные запросы, командные файлы и т.д.)

2. Создание БД.

2.1. Проектирование структуры БД.

2.2. Создание файла БД.

2.3. Заполнение файла БД записями.

3. Создание комплекса вспомогательных сервисных файлов.

3.1. Файл формата экрана.

3.2. Файлы выходных форм документов.

3.3. Индексированные файлы.

3.4. Файлы стандартных запросов.

3.5. Комплекс командных файлов.

4. Эксплуатация созданной ИС.

4.1. Поддержка информации в БД.

4.2. Получение необходимой информации через файлы запросов.

Согласно этим этапам разработаем прикладную программу, которая иллюстрирует возможности СУБД в автоматизации управления школой.

Постановка задачи. Цель разработки этого программного средства - автоматизация работы завуча средней школы по одной из форм отчета всеобуча.

Предполагается получение информации в виде табл. 6.3 и табл. 6.4.

Таблица 6.3
Информация для отчета завуча школы

Класс	Сохранение контингента		Неуспевающие	Второгодники, их успеваем.	Пропуски		Работа с трудными учащимися и их семьями	
	Конец года	Четверти 1 2 3 4	Четверти 1 2 3 4	Четверти 1 2 3 4	Без ув. причин	По болезни	Состоящ. на учете	Трудные семьи

[illegible]

Входной информацией в БД выступают следующие данные по каждому классу: число учеников на конец прошлого года и по четвертям; число неуспевающих в классе по четвертям; число пропусков в классе на текущий момент (по болезни и без уважительной причины. Учитывается работа с трудными семьями, медицинское обслуживание. Собирается информация о сохранности учебников и наглядно-дидактических материалов; о группах продленного дня и посещении УПК.

Требования к переработке информации:

- ввод данных и автоматическая сортировка по классам;
- добавление записей;
- внесение изменений в записи по запросу на номер класса;
- удаление записей посредством пометки знаком «*» в режиме просмотра данных. Выход-

ная информация:

- просмотр данных в виде табл. 6.3 или табл. 6.4, а также просмотр данных по одному из столбцов таблиц, например, только пропуски уроков;
- вывод на принтер табл. 6.3 или табл. 6.4, а также печать только определенных столбцов.

- вывод на принтер табл. 6.3 или табл. 6.4, а также печать только определенных столбцов.

Ввод данных удобно осуществлять с использованием экранных форм, повторяющих на экране таблицы. Так как эти таблицы не помещаются на экране одновременно и содержат неповторяющуюся информацию, целесообразно разделить ввод данных на две части. Обработку информации также удобнее проводить по двум таблицам по отдельности. Выбор осуществим по меню:

1. Часть 1. 2. Часть 2. 3. Конец работы.

Создадим две БД и проиндексируем их с тем, чтобы при вводе данных они автоматически сортировались по классам и чтобы в дальнейшем ускорить поиск требуемой информации. Над каждой из двух таблиц организуем проведение следующих операций:

- 1) заполнение новой базы;
- 2) добавление записей;
- 3) корректировку;
- 4) удаление записей;
- 5) печать документов;
- 6) выход.

Рассмотрим характеристики программ по каждому из представленных пунктов меню.

1) Этот пункт предполагает, что все записи БД уничтожаются и предварительно появляется предупреждение:

«При создании новой БД все данные уничтожаются. Вы действительно хотите создать новую БД? (д/н)».

При нажатии клавиши «д» пользователю предоставляется пустая экранная форма, соответствующая выбранной таблице из главного меню.

2) При добавлении записей на экране высветит пустую экранную форму. Пользователь может вносить информацию, передвигаясь по столбцам. При этом курсор автоматически по заполнении столбца переходит в следующий. После заполнения каждой строки в нижней части экрана высвечивается сообщение:

«Продолжить добавление записей? (д/н) :: »

При ответе «н» происходит возврат в меню, а записи сортируются по классам.

3) Для корректировки также высветим экранную форму, но с уже заполненными столбцами. Здесь необходимо предусмотреть тот случай, когда изменения должны вноситься только в отдельные классы. В этой связи требуется создание подменю.

4) При удалении записей, необходимо, как и в режиме корректировки, предусмотреть подменю с запросами по классам. Пользователю необходимо поставить знак «звездочка» (*) в столбце «классы», о чем в нижней части экрана организуем подсказку:

«Для удаления записи, на которой стоит курсор, пометьте ее звездочкой (*)»

По завершении работы все помеченные записи уничтожаются, а оставшиеся автоматически сортируются по классам.

5) Этот раздел служит для получения выходной информации. При его выборе создадим еще одно меню:

(Часть 1)		
1) Полный отчет	2) Сохранение контингента	3) Неудачающие
4) Успешность вторых годовников	5) Пропуски	6) Трудные учащиеся

(Часть 2)		
1) Полный отчет	2) Многодетные семьи и опекаемые дети	
3) Горячее питание	4) Медицинское обслуживание	
5) Сохранность учебников	6) ГПД	7) УПК

Здесь по требованию пользователя должна выводиться информация либо о всех классах сразу, либо только о тех, которые укажет пользователь в ответе на запрос В этом же запросе сразу определим, на какое устройство осуществляется вывод. Вид запроса:

«Какие классы Печать (д/н)»

Полный отчет (первый пункт меню) выводится в соответствии с экранными формами табл. 6.3 и табл. 6.4. Для всех остальных пунктов создадим форматные файлы, которые выводят информацию по одному из столбцов экранных форм.

Таблица 6.5
Отчет по «Медицинскому обслуживанию»

Медицинское обслуживание			
Класс	Количество осмотров	Хронические заболевания	Специальные медицинские группы
1а	30	3	1
1б	26	2	0
1в	30	0	0
2а	29	1	1
...	...	...	...
** итог **	820	21	14

Структура БД. Создадим две БД со структурами, изображенными в табл. 6.6. Поле «класс» содержит информацию о номере класса. По полю «класс1» различаются параллельные классы.

Для быстрого поиска информации в базе, а также для автоматической сортировки записей после внесения изменений, проиндексируем их по полю «класс» и «класс1». При этом создаются

Таблица 6.6
Структуры создаваемых баз данных

Поле	Имя	Тип	Длина
001	класс	N	002
002	Класс1	C	002
003	пр:год	N	005
004	четв:1	N	002
005	четв:2	N	002
006	четв:3	N	002
007	четв:4	N	002
008	неусп:ч1	N	002
009	неусп:ч2	N	002
010	неусп:ч3	N	002
011	неусп:ч4	N	002
012	втгод:ч1	N	002
013	втгод:ч2	N	002
014	втгод:ч3	N	002
015	втгод:ч4	N	002
016	проп.без у/п	N	004
017	по:болезни	N	004
018	на:учете	N	005
019	тр.семья	N	005

Поле	Имя	Тип	Длина
001	класс	N	002
002	класс1	C	002
003	семьи	N	005
004	дети	N	005
005	фонд	N	005
006	молоко	N	004
007	пол1	N	003
008	пол2	N	003
009	осмотр	N	005
010	хрон	N	005
011	спрг	N	004
012	учебн	C	010
013	гпд	C	003
014	упк	C	005

Создадим форматные файлы отчетов для вывода информации по одному из столбцов таблиц: PROP.FRM (пропуски), TRUD.FRM (трудные учащиеся), UPK.FRM (УПК), SOHR.FRM (сохранение контингента), WTORG.FRM (успеваемость второгодников), NEUSP.FRM (неуспевающие), SEM.FRM (многодетные семьи и опекаемые), PIT.FRM (горячее питание), MED.FRM (медицинское обслуживание), UBN.FRM (сохранность учебников), GRD.FRM (группы продленного дня).

Ниже представлены тексты некоторых программ, поддерживающие рассматриваемую информационную систему (автор -Т.П. Глазко).

Программа 140

```

_____ модуль MAIN. PRG _____
erase
set colon on
set talk off
store ' ' to r1,dd,d3
do while d3<>'3'
erase
do while dd<>'1' .and. dd <>'2' .and. dd<>'3'
@ 5,12 say ' 1). Часть 1 '
@ 6,12 say ' 2). Часть 2
@ 7,12 say ' 3). Конец работы '
@ 8,12 say ' Введите цифру ' get dd
read
loop
enddo
If dd<>'3'
do case
case dd= '1'
use newws Index k1
case dd= '2'
use newws1 Index k11
endcase
do menu
else
store '3' to d3
endif store ' ' to dd
loop
enddo
cancel
_____ МОДУЛЬ MENU. PRG _____

```

MENU.PRГ поддерживает меню операций и в зависимости от выбранного пункта передает управление модулям: SERVIS2.PRГ (корректировка), APPN PRГ (добавление), NEWBD.PRГ (создание новой БД), SERVIS3.PRГ (удаление), DOC.PRГ (документы). Прежде чем будет вызвана программа SERVIS2.PRГ, выполняется еще одна программа - DOP.PRГ, которая к этому меню добавляет запрос на классы.

Программа 141

```

set format to screen
store 0 to d1, d2
do while d2<>8
goto top
erase
store 0 to d1
text
      МЕНЮ
      1) Заполнение новой базы      4) Удаление записей
      2) Добавление записей        5) Документы
      3) Корректировка             6) Выход

endtext
do while d1<>1 .and. d1<>2 .and. d1<>3 .and. d1<>4; .and. d1<>5 .and.
d1<>6
@ 9,1 say ' Введите цифру' get d1 picture '9'
read
loop
enddo
do case
case d1=3

```

```

do DOP
do SERVIS2
case dl=2
do APPN
case dl=1
do NEWBD
case dl=4
do SERVIS3
case dl=5
do DOC
case dl=6
store 8 to d2
endcase
loop
enddo
return
_____ модуль DOP.PRG _____

```

DOP.PRG осуществляет запрос на классы, по результатам ввода формирует логическую переменную kkk. используемую в некоторых других модулях для контроля за номерами классов, kkk принимает значение «истина», если поле «класс» соответствует введенным номерам, и «ложь» - в противоположном случае.

Программа 142

```

store 0 to kl
store to kl
store T to pp
*** запрос на классы
@ 10,1 say 'Какие классы'
do while pp
@ 10,13 get kl picture '99-99'
read
if val($(kl,1,2))<12 .and. val($(kl,4,2))<12
store F to pp
endif
loop
enddo
*** Формирование логической переменной kkk
if val ($(kl,4,2))=0
store 12 to kl
else
store val($(kl,4,2))+1 to kl
endif
store 'Класс>val ($(kl, 1,2))-1 .and. Knacc<kl' to kkk
return
_____ модуль SERV1S2.PRG _____

```

Этот модуль позволяет внести изменения в записи БД. Классы, в которых будет производиться корректировка, пользователь указывает, отвечая на запрос до вызова настоящей программы.

Программа 143

```

erase
set colon off
set talk off
goto top
    if val(dd)=1
    do shapka
    else do shapkal

```

```

endif
    store 1 to num1 ,num2
    store T to log
    store ' ' to r1
    store 0 to pointer, num
    store 't' to dl
count to num
do while num1<num .and. num2<num .and. log
store 10 to line
store 't' to dl
    @ 21,25 say 'Нажмите<к>> для перехода к другим записям
    @ 22,25 say "или ВВОД для внесения изменений"
if val(dd)=1
do newws2
else
do newws21
endif
skip - line + 10
store 10 to line
    do i zm2
store pointer+1 to pointer
loop
if num1<num
skip
endif
enddo
_____ модуль SHAPKA.PRG _____

```

Программа 144

```

erase
text
таблица 1

```

Класс	Сохранение контингента		Неуспевающие	Второгодники, их успеваем.	Пропуски	Работа с трудными учащимися и их семьями		
	Конец года	Четверти 1 2 3 4	Четверти 1 2 3 4	Четверти 1 2 3 4	Без ув. причин	По болезн	Состоящ. на учете	Трудные семьи

```

endtext

```

```

return .

```

_____ модуль SHAPKA1.PRG _____

Программа 145

```

erase
text
таблица 2

```

Класс	Многодетные семьи и опекаемые дети	Горячее питание	Медицинское обслуживание	Учебники, ГПД	УПК
				дидактические мате-	

семьи	опекаемые	фонд	б/п	1	2	Количе-	Хрониче-	риалы
	дети	все-	мо-	п.	п.	ство	ские забо-	
		обуча	локо			осмотров	левания	

Контрольные вопросы и задания

1. Как сформировалось понятие «база данных»? Какое место базы данных занимают в структуре информационных систем?
2. В чем состоит концепция ANSI/SPARC трехуровневого представления данных?
3. Что означает «концептуальное представление данных»?
4. Из каких этапов складывается проектирование баз данных?
5. Какие принципы системного анализа используются при проектировании баз данных?
6. Охарактеризуйте основные параметры СУБД типа dBASE.
7. Каковы основные компоненты СУБД dBASE IV?
8. Каковы назначение и основные функции управляющего центра?
9. Файлы каких типов используются в системе dBASE?
10. Как происходит создание файла БД?
11. Охарактеризуйте возможности ввода информации в файл БД.
12. Как формируются запросы к БД?
13. Как формируются экранные формы?
14. Как происходит разработка отчетов?
15. Каков синтаксис команд командного языка dBASE?
16. Запишите команды и поясните назначение опций:
 - а) создания файла БД;
 - б) открытия файла БД;
 - в) создания индексного файла;
 - г) открытия индексного файла;
 - д) вызова экранных форм, редактирования и ввода данных;
 - е) просмотра информации;
 - ж) перемещения по файлу;
 - з) вставки и удаления записей;
 - и) сортировки и поиска;
 - к) создания экранных форм ввода-вывода;
 - л) оформления экранного интерфейса.
17. Запишите дополнительные команды ввода-вывода.
18. Какие команды используются для создания форм отчетов и печати отчетов?
19. Какие команды используются при создании программных файлов dBASE?
20. Как оформляются и вызываются программные процедуры?
21. Какие возможности описания и начального присвоения переменных имеются в языке dBASE?
22. Какие команды позволяют организовать диалоговые меню программ?
23. Какой вид имеют управляющие конструкции ветвления, выбора и цикла в языке dBASE?

§ 3. АВТОМАТИЗИРОВАННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

3.1. АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ

Во второй половине 60-х и в 70-х годах получили развитие, так называемые, автоматизированные системы управления сложными объектами хозяйственной деятельности (предприятиями, энергосистемами, отраслями, сложными участками производства).

Автоматизированная система управления (АСУ) - это комплекс технических и программных средств, совместно с организационными структурами (отдельными людьми или коллективом), обеспечивающий управление объектом (комплексом) в производственной, научной или общественной среде.

Цель разработки и внедрения АСУ - улучшение качества управления системами различных видов, которое достигается

- своевременным предоставлением с помощью АСУ полной и достоверной информации управленческому персоналу для принятия решений;
- применением математических методов и моделей для принятия оптимальных решений.

Кроме того, внедрение АСУ обычно приводит к совершенствованию организационных структур и методов управления, более гибкой регламентации документооборота и процедур управления, упорядочению использования и создания нормативов, совершенствованию организации производства. АСУ различают по выполняемым функциям и возможностям информационного сервиса.

АСУ подразделяют по функциям:

- административно-организационные (например системы управления предприятием — АСУП), отраслевые системы управления - ОАСУ);
- технологические (автоматизированные системы управления технологическими процессами - АСУТП, в свою очередь подразделяющиеся на гибкие производственные системы - ГПС, системы контроля качества продукции - АСК, системы управления станками и линиями с числовым программным управлением);
- интегрированные, объединяющие функции перечисленных АСУ в различных комбинациях.

По возможностям информационного сервиса различают информационные АСУ, информационно-советующие, управляющие, самонастраивающиеся и самообучающиеся.

Первоначально АСУ строились на основе больших ЭВМ, имевшихся в вычислительных центрах крупных предприятий и организаций, и предполагали централизованную обработку информации. Помимо штата вычислительного центра обслуживание АСУ требовало создания специального подразделения численностью 200 -300 человек.

С появлением персональных компьютеров (ПК) и локальных вычислительных сетей (ЛВС) основой программно-аппаратного обеспечения АСУ стали распределенные информационные системы в сети ПК с архитектурой клиент - сервер. Такие системы позволяют вести учет событий и документальных форм по месту их возникновения, полностью автоматизировать передачу информации лицам, ответственным за принятие решений, создавая, таким образом, предпосылки для перехода к безбумажным комплексным технологиям управления, охватывающим все участки и подразделения предприятий и учреждений, весь производственный цикл.

Остановимся подробнее на структуре и функциях АСУП - наиболее распространенной и одновременно наиболее сложной разновидности АСУ. Управление производством - сложный процесс, требующий согласованной деятельности конструкторов, технологов, производственников, экономистов, специалистов по снабжению и сбыту.

В задачи управления входят

- разработка новых изделий;
- определение технологий изготовления изделий, проектирование оснастки;
- расчет пропускной способности оборудования, потребностей во всех видах ресурсов и производственной программы (плана);
- учет процесса производства, контроль за расходом комплектующих, сырья, ресурсов;
- расчет издержек производства и основных технико-экономических показателей (прибыли, рентабельности, себестоимости и др.).

Многие задачи, с которыми приходится сталкиваться АСУП, оказываются не поддающимися четкой формулировке, их решение основывается на неформальных факторах (например, социально-психологический климат, стиль руководства).

Цели внедрения любой АСУП:

- повышение эффективности принимаемых решений, особенно в части наилучшего использования всех видов ресурсов и сокращения потерь, достигаемых за счет обеспечения процесса принятия решений своевременной, полной и точной информацией, а также применения математических методов оптимизации;
- повышение производительности труда инженерно-технического и управленческого персонала (и его сокращение) за счет выполнения основного объема учетных и расчетных задач на ЭВМ.

Независимо от профиля АСУП они обладают однотипной функциональной структурой, рис. 6.8.



Рис. 6.8. Функциональная структура АСУП

Блок 1 - источники информации. В их роли могут выступать учетчики на различных участках производства, снабжения и сбыта, датчики на рабочих местах. Среди источников информации могут быть и внешние, такие как заказы на поставку продукции, нормативные акты, информация о ценах и другая документация.

Блок 2 выполняет предварительную обработку данных (проверку и уточнение), а затем передает ее в базу данных (блок 3) или непосредственно для последующей обработки и анализа (блок 4).

Блок 3 - база или банк данных. Данные являются результатом сбора информации, измерений характеристик объектов и процессов управления и в таких системах представляются в соответствии с определенными стандартами, образуя базу данных.

Блок 4 обработки и анализа информации - центральный блок АСУ. Он решает следующие задачи:

- управления базой данных, в том числе обеспечения ее обновления и целостности, защиты от несанкционированного доступа;
- реагирования в непредвиденных и аварийных ситуациях, требующих быстрого решения;
- финансовых и учетно-бухгалтерских расчетов типа учета состояния фондов, финансовых и налоговых операций, расчета прибыли и рентабельности;
- составления календарных и оперативных планов, обеспечения заказов на материалы и комплектующие, контроля за выполнением договоров, управления сбытом готовой продукции;
- оценки и прогнозирования рынка, анализа работы трудового коллектива;
- проектно-технологических расчетов.

Важнейшее значение при обработке и анализе информации играют экономико-математические модели.

С точки зрения общей организации управления можно выделить следующие основные группы практически используемых экономико-математических моделей:

- а) прогнозирования показателей развития предприятия или объединения;
- б) оптимизации производственной программы предприятий или объединений;
- в) распределения производственной программы по календарным периодам;
- г) оптимизации направлений использования фонда развития предприятия или объединения;
- д) оптимизации внутрипроизводственных транспортных потоков;
- е) оптимизации использования отдельных видов ресурсов;
- ж) оптимизации всякого рода нормативов ведения производственно-хозяйственной деятельности предприятий или объединений (партий деталей, норм запасов, размеров производственных резервов и т.д.);
- з) разработки балансов производственно-хозяйственной деятельности.

Прогнозирование показателей развития предприятия или объединении осуществляется на основе пользования, главным образом, методов математической статистики. Последние позволяют ориентировочно определить тенденции изменения в перспективе показателей объема выпуска продукции, ее трудоемкости, величины затрат на производство и т.д. Как правило, для использования подобных методов необходимы статистические сведения о деятельности предприятия или объединения в прошлом.

Для определения тенденций развития производственно-хозяйственной деятельности на относительно близкую перспективу используют всякого рода экстраполяционные методы. Для этих

целей на основе статистических сведений за прошедшие периоды рассчитывают соответствующие тенденциям развития того или иного аспекта производственно-хозяйственной деятельности регрессионные показатели, которые впоследствии применяют для оценки вероятных перспективных направлений.

Решение задач оптимизации производственной программы сводится к формированию таких номенклатур и объемов выпуска продукции, которые в условиях наличных и выделяемых ресурсов, контрольных показателей потребности рынка и ведения деятельности обеспечивали бы оптимизацию принятого критерия. Для решения задач такого класса широко применяют разнообразные модели, базирующиеся на методах линейного программирования; при этом в качестве исходных данных требуются контрольные показатели по выпуску продукции, величина ресурсов (труда, машинного времени и материалов), а также нормы расхода исходных ресурсов на изготовление единицы продукции.

Распределение производственной программы по календарным периодам выражается в установлении номенклатуры и объема выпуска продукции в определенные месяцы и кварталы года. Основной задачей использования моделей такого класса является обеспечение стабильности производственно-хозяйственной деятельности объединения или предприятия в течение рассматриваемого периода.

Оптимизация направлений использования фонда развития производства характерна для объединений, включающих значительное число предприятий. Решение этой задачи позволяет определить рациональные пути использования фонда развития, обеспечивающие достижение оптимума какого-либо критерия (максимизации выпуска продукции, минимизации затрат на производство или максимизации прибыли и т.д.).

Оптимизация использования отдельных видов ресурсов может осуществляться на самых различных уровнях управления производством. К данному классу задач можно отнести оптимизацию раскроя материалов, образования разнообразных смесей, использования оборудования, распределения работ по линиям и т.д. Наиболее типичным представителем данного класса задач является задача образования смеси из разнообразных исходных компонентов с целью минимизации затрат на производство. Такие задачи имеют место практически во всех отраслях народного хозяйства (от нефтепереработки до производства мороженого).

Разработка балансов производственно-хозяйственной деятельности предприятий или объединений осуществляется на основе использования математического аппарата межотраслевого баланса производства и распределения продукции.

Блок 5 - система формирования выходной информации - обеспечивает подготовку (обычно в печатном виде) различного рода сводок, справок, форм, технологических карт, чертежей и проектной документации, необходимых на производственных участках.

Автоматизированная система управления предприятием может состоять из следующих подсистем управления:

- технической подготовки производства (конструкторской и технологической подготовки);
- технико-экономического планирования;
- бухгалтерского учета;
- управления материально-техническим снабжением;
- оперативного управления основным и вспомогательными производствами;
- управления сбытом;
- управления кадрами;
- управления качеством;
- управления финансами;
- нормативного хозяйства и др.

Необходимо отметить, что реализация многих проектов АСУП в 70-е годы в нашей стране и во всем мире закончилась неудачей - эти системы «не прижились», оказались нежизнеспособными. В первую очередь, это вызвано тем, что в их концепции были заложены претензии на слишком высокую степень автоматизации управления, не оставляющую места для человека-руководителя. Кроме того, многие математические модели в АСУП были недостаточно точными и приводили к ошибкам.

3.2. ИНФОРМАЦИОННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ

3.2.1. Общие принципы

В ходе развития концепции АСУ с целью преодоления указанных выше трудностей была выработана новая концепция компьютеризации и автоматизации управленческой деятельности, которая уже не ставит столь глобальных задач, а направлена прежде всего на автоматизацию обработки документов в системах управления организациями и предприятиями - документооборота.

Известно, что в практике управления по-прежнему преобладают бумажные потоки документации, которые в современных условиях становятся все менее и менее эффективными. Решение задач учета и контроля, сведения баланса и распределения ресурсов, планирование мероприятий выполняется по старинке, сохраняя почву для многочисленного управленческого аппарата, для бюрократизма.

Задача компьютеризации управленческой деятельности в различных отраслях первоначально характеризовалась, в основном, массовой разработкой и применением прикладных программ персональных компьютеров, автоматизирующих рабочие места (АРМ) управленческого персонала ранга секретарей и референтов руководителей, начальников отделов и служб учреждений.

Данный подход к использованию компьютеров оказался экономически не вполне оправданным. Помимо затрат на приобретение электронно-вычислительной техники, он привел к разрастанию вспомогательного управленческого персонала высокой квалификации.

Для повышения эффективности применения компьютеров в управлении необходимо переосмысление принципов организационного управления предприятиями и организациями, представление их в виде технологий обработки данных и информационных потоков, а также взаимная адаптация «живой» и компьютерной систем управления. Акцент автоматизированного накопления и обработки информации перемещается на передачу информации средствами компьютерных телекоммуникаций» что ведет к обществу безбумажной информатики. С точки зрения практики, это означает, с одной стороны, создание и развитие глобальных компьютерных сетей, что требует большой работы по унификации и стандартизации самих форм и содержания управленческой деятельности даже в пределах отдельных отраслей, а с другой стороны - создание локальных вычислительных сетей в пределах каждого учреждения.

Наибольшие проблемы систем информационного обеспечения управления связаны с общей эффективностью, жизнеспособностью и способностью интегрироваться с существующие «человеческие» системы управления. Практика реализации многих таких проектов говорит о том, что информационные системы управления часто не выходят за рамки экспериментальных разработок, демонстрационных и прототипных моделей, отторгаются традиционной системой управления и оказываются нежизнеспособными. Это связано как с недостаточными подготовкой и заинтересованностью управленческого персонала в использовании компьютерных технологий, дефектами сложившейся технологии управления, так и с несовершенством систем, создаваемых на данном этапе, с искусственностью многих функций таких систем информационной поддержки управления, несоответствием их требованиям пользователей, т.е. недостатками концептуальной модели.

Большинство задач управления допускают следующую декомпозицию:

1) задача учета и контроля событий по месту их возникновения;

2) баланс и распределение материальных ресурсов;

3) планирование мероприятий и управление временными ресурсами. В сфере управления предприятием можно условно выделить объективную и субъективную стороны. Объективная сторона управления предприятием зафиксирована в его документообороте и включает некоторый набор документальных форм, а также инструктивно зафиксированные правила заполнения, обращения и хранения этих форм.

При анализе документооборота учреждения в целях проектирования информационной системы управления должны быть определены и зафиксированы

- название (идентификатор) формы;
- назначение формы (указание на то, какие решения данная форма позволяет принимать);
- условия или ситуация, в которой форма заполняется;
- на основе каких источников составляется;

- откуда форма поступает и куда передается, а также условия ее хранения (место и срок).

Анализ документальных форм позволяет значительно рационализировать документооборот предприятия. В первую очередь, это означает отказ от тех форм, которые

- 1) дублируют друг друга;
- 2) не используются при принятии управленческих решений или не преобразуются в другие формы;
- 3) преобразуются в такие формы, которые не используются при дальнейшей обработке.

Однако, объективная сторона деятельности предприятия не позволяет получить в полной мере его адекватную информационную модель. Подчас более важным для жизнеспособности будущей информационной системы управления оказывается анализ субъективной стороны деятельности предприятия, выявляющий

- распределение обязанностей и функций, сферы компетентности управленческих работников при принятии решений;
- приоритеты в процессе обработки документальных форм, на которых основывается принятие решений;
- документально не зафиксированные тонкости управления штатом.

В процессе извлечения необходимой информации у руководящих работников предприятия необходимо исходить из того, что они не знакомы с технологией проектирования баз данных, понятийным аппаратом информатики и не обладают знаниями в области компьютерной техники. Процедура извлечения знаний у руководящих работников предприятия включает в себя составление информационной схемы предприятия, включающей обозначения отделов и подразделений, а также указание информационных связей между ними, выявление объектов, на которые направлена деятельность организации и ее подразделений, и т.д.

Технической базой системной компьютеризации управления может выступать локальная сеть персональных компьютеров, однако лишь при условии создания распределенной базы данных предприятия или учреждения. Данный подход позволяет

- производить учет событий на месте их возникновения в естественной форме в распределенной базе данных без дублирования отчетных форм;
- устранять необходимость отслеживания причинно-следственных связей и временных зависимостей при решении множества взаимосвязанных задач;
- повышать устойчивость системы управления против случайного или намеренного искажения информации;
- обеспечивать доступ руководства к первичным данным и повышать общий уровень его компетентности при принятии решений (и тем самым делает ненужным многочисленный управленческий аппарат).

Важным условием применения такого подхода является развитие клиент-серверной архитектуры локальных сетей вычислительных машин и преодоление ограниченности существующих архитектур баз данных.

Среди функций СУБД, отвечающих за взаимодействие пользователя и машинной системы и пользователей между собой, выделяются две основные.

1. Защита информации и разграничение доступа пользователей к ней. При использовании информации базы данных обычно имеется некоторое столкновение интересов пользователей, которое может привести к уничтожению или искажению информации, к несанкционированному ее распространению и использованию. Некорректные действия отдельных пользователей могут нанести ущерб остальным пользователям и базе данных в целом. Чтобы избежать этого, в СУБД имеются средства разграничения доступа пользователей и другие средства защиты информации. Разделы базы данных могут быть закрыты для пользователя совсем, открыты только для чтения или открыты для изменения. Кроме того, при многопользовательском режиме работы с базой данных, когда с данными одновременно работают несколько пользователей (и вносят в них изменения), необходимо, чтобы изменения корректно вносились в базу данных (сохранялась целостность данных). Для сохранения целостности данных служит механизм транзакций при манипулировании данными - выполнение манипуляций небольшими пакетами, результаты каждого из которых в случае возникновения некорректности операций «откатываются» и данные возвращаются к исходному состоянию.

2. Интерфейс с пользователями, который обеспечивается средствами ведения диалога. По

мере развития и совершенствования СУБД этот интерфейс становится все более и более дружелюбным. В перспективе средства ведения диалога пользователя с СУБД должны приобрести интеллектуальные свойства и обеспечить возможность ведения диалога на естественном языке.

Сформировавшееся к настоящему времени программное обеспечение информационной поддержки управления настолько многообразно, что способно удовлетворять потребности пользователей самых различных категорий - от случайного пользователя, для которого обращение к рассматриваемой среде - единичный эпизод, до профессионала высокой квалификации в области разработки систем баз данных.

Одна из возможных реализации информационной системы управления предусматривает наличие двух основных компонентов - SQL-сервера и дружелюбных интерфейсов конечных пользователей. Узлами сети являются компьютеры или рабочие станции, установленные в кабинетах руководителя предприятия и его заместителей, в отделе кадров, бухгалтерии, других структурных подразделениях.

В качестве примера объектной распределенной СУБД, которая может быть использована для автоматизации системы управления, рассмотрим систему SQLWindows, первоначально разработанную компанией «Gupta Technologies», а позже развитую «Microsoft». Данная система позволяет работать в сети с наиболее популярными SQL-серверами. Пакет SQLWindows отличается от аналогичных систем наиболее полной реализацией возможностей, необходимых для эффективной объектно-ориентированной разработки сетевых прикладных программ. В первую очередь, к его преимуществам можно отнести

- высокую автоматизацию процесса разработки;
- поддержку ручного программирования;
- развитые средства объектно-ориентированного программирования;
- встроенные средства поддержки коллективной разработки.

Данная система обладает многими возможностями, необходимыми для создания SQL-приложений в среде Windows, например, развитым графическим интерфейсом, средствами построения отчетов, отображения информации в графической форме и т.п.

SQLWindows обладает встроенным языком, необходимым для разработки сложных прикладных программ. Этот язык носит название SAL (SQLWindows Application Language). Он не похож ни на какие другие языки и по своему синтаксису и семантике может быть отнесен к языкам четвертого поколения. Его синтаксис сильно упрощен, он содержит только двенадцать ключевых слов, но при этом обладает большой эффективностью и выразительностью. Так например, в SQLWindows операторные скобки (типа begin...end в Паскале) реализованы посредством сдвига кода программы вправо. Этот сдвиг формируется автоматически при переходе на следующий уровень вложенности. Редактор программы для языка SAL является контекстно-чувствительным.

Это означает, что на экране постоянно имеется список лексем и идентификаторов, которые могут быть использованы в текущем контексте. При изменении положения курсора этот список автоматически изменяется, обеспечивая постоянную подсказку пользователю при написании программ. SAL является полностью объектно-ориентированным языком.

SQLWindows обеспечивает

- полиморфизм (polymorphism);
- скрытые логики обработки (encapsulation);
- определяемые пользователем типы (user defined types);
- наследование содержимого окна (window contents inheritance);
- множественное наследование (multiple inheritance).

Все классы объектов в SQLWindows делятся на графические и функциональные. Графические классы служат для отображения информации, функциональные - для создания переменных или подпрограмм, которые могут использоваться в графических классах. Посредством множественного наследования переменные или подпрограммы из функционального класса могут быть наследованы другими классами.

В SQLWindows существует три типа объектов: объекты-родители, или объекты высокого уровня (Top-level Objects), объекты-наследники (Child Objects) и меню. К объектам высокого уровня относятся диалоговые окна, экранные формы для ввода и просмотра данных, окна работы с таблицами и т.д. Объекты-наследники являются элементами объектов высокого уровня. К ним относятся тексты заголовков, меток и инструкций, списки, поля ввода, «кнопки», рамки, графиче-

ские изображения и т.п. Меню также являются объектами-наследниками, но создаются они особым способом. Поэтому они образуют отдельный тип. Возможно создание как одиночного меню, так и каскада последовательно открывающихся меню.

Для облегчения труда разработчиков имеется набор заранее созданных заготовок, так называемых «быстрых объектов» (Quick Objects). Эти объекты позволяют быстро создавать прикладные программы, отображать информацию в виде таблиц, полей данных и других графических элементов, строить окна и меню, задавать связи между таблицами. При этом не требуется создавать прикладные программы вручную, хотя при желании это возможно. В системе SQLWindows имеются также специальные быстрые объекты для обработки данных из некоторых систем электронных почт. Например, с помощью SQLWindows можно создать проблемно-ориентированную клиентскую часть, которая сохраняет информацию, получаемую по электронной почте, в базе данных. Получаемые данные доступны не только для чтения, но и для коррекции. Разработчик может строить собственные объекты и использовать их наравне с Quick Objects. Кроме того, он может создавать новые классы объектов на основе уже существующих, в том числе и на основе классов и объектов из набора Quick Objects. Таким образом Quick Objects не только обеспечивает быстрое создание приложений в SQLWindows, но и служит базой для построения новых объектов.

В тех случаях, когда разработка приложения ведется совместно несколькими участниками в сети ЭВМ, используется среда TeamWindows, содержащая набор средств для управления коллективной разработкой проекта, создания исходного кода приложения и разработки экранов SQLWindows в многопользовательском режиме. Средствами TeamWindows осуществляется контроль версий, протоколирование изменений компонентов приложений, ролевое разграничение доступа к его компонентам, выпуск отчетов о ходе разработки, поддержка стандартов кодирования, создание схемы приложений и многое другое. Среда TeamWindows состоит из нескольких компонентов.

1) Система управления разработкой проекта (TeamWindows Project Development Manager) - наиболее важный компонент TeamWindows; чаще всего именно ее называют TeamWindows. Данный компонент содержит средства для управления проектами SQLWindows, разработки приложений, поддержки стандартных библиотек и создания экранных форм.

2) Система управления словарем данных (Data Dictionary Manager) содержит средства для управления информацией о базе данных создаваемого приложения, а также поддерживает некоторые административные задачи, например, позволяет хранить данные о коллективе разработчиков. Словарь данных содержит информацию о структуре баз данных разрабатываемого приложения: названия таблиц и колонок, тип отношения между данными и т.п. Словарь данных является частью репозитория данных.

3) Репозиторий данных (Data Repository) - централизованная многопользовательская база данных, хранящая все компоненты создаваемой проблемно-ориентированной клиентской части и всю информацию о ней, включая содержимое всех экранов и текущую копию каждого модуля проекта. Под модулем понимается любой относящийся к проекту файл. В репозитории собрана также вся информация об участниках проекта, их правах доступа к его компонентам, стандартах кодирования и т.п.

4) Библиотеки стандартных программ (Template Libraries). Каждая библиотека поддерживает набор экранных классов и общих функций, которые позволяют легко строить приложения SQLWindows.

Таким образом, пакет SQLWindows является подходящей основой для создания объектно-ориентированной информационно-поисковой системы для управления учреждениями различного профиля.

К информационным системам рассматриваемого типа можно отнести и некоторые другие программные продукты более прикладного характера, такие, например, как Lotus Notes - систему автоматизации документооборота учреждения. В России большую известность получили системы автоматизации бухгалтерского учета - продукты компаний «1С», «Парус», «Галактика» и т.д.

Компоненты этих систем, аппаратно реализованные на базе персональных компьютеров, размещенных на рабочих местах и объединенных в корпоративную локальную сеть, в последнее время часто представляют собой сервер (или несколько серверов), на котором хранятся базы данных и набор клиентских частей, представляющих собой автоматизированные рабочие места (АРМ) работников определенных служб предприятия.

3.2.2. Информационные системы управления в образовании

В отечественной системе образования первые информационные системы управления создавались еще в 60-е годы.

Можно выделить следующие уровни управленческой деятельности с использованием ЭВМ в системе образования:

- 1) управление обучением и развитием отдельного учащегося;
- 2) управление учебным процессом в рамках одного учебного заведения;
- 3) управление работой группы родственных учебных заведений;
- 4) управление учебными заведениями по территориальному принципу;
- 5) управление системой народного образования страны.

На первом уровне задачи управления совпадают в значительной мере с задачами обучения с помощью компьютеров; этим вопросам посвящен параграф 5 данной главы.

На втором уровне реальные успехи достигнуты прежде всего в вузах. С одной стороны, государственное высшее учебное заведение достаточно велико по контингенту учащихся и преподавателей и имеет достаточно большую материальную базу для того, чтобы использование компьютеров в управлении было экономически оправдано, с другой - в вузах, особенно технических, наличествуют достаточно профессионально подготовленные кадры для решения проблемы информатизации управления. При этом преследуются следующие цели:

- повышение качества подготовки специалистов за счет совершенствования управления со стороны ректората, деканатов, кафедр;
- повышение качества учебной, учебно-методической, научно-исследовательской деятельности на основе оперативной информации;
- повышение эффективности в разработке учебных планов и программ, составлении расписания занятий, других видов аудиторной и внеаудиторной работы.

Традиционными программными подсистемами информационной системы управления вузом являются *Абитуриент, Кадры, Учебные планы и программы, Зарплата, Стипендии, Текущая успеваемость. Нагрузки преподавателей. Сессия* и другие. Подобные программы используются в большинстве вузов России.

Вместе с тем, эти подсистемы редко образуют единую информационную систему управления. Незрелость информационной среды, отсутствие в большинстве вузов полноохватной локальной сети, материальные трудности, неподготовленность управленческого персонала и другие факторы препятствуют созданию систем типа «клиент - сервер» с единым администрированием, гарантией отсутствия противоречивых данных, защитой целостности и конфиденциальности данных.

Что же касается построения современных информационных систем управления в образовании на территориальном уровне и в масштабах страны в целом, то эта задача является актуальной и находится в стадии решения. В Национальном докладе России на II Международном конгрессе ЮНЕСКО «Образование и информатика», проходившем в Москве в июле 1996 г., говорится:

«В рамках реформы системы образования России идет и реформа управления системой, поиск наиболее рациональных соотношений централизации и децентрализации управления...

Для управления качеством учебного процесса создаются информационные системы мониторинга и государственных образовательных стандартов.

Информатизация образования требует весьма значительных материальных и финансовых ресурсов, сравнимых по объему с годовым национальным доходом страны. Поэтому в России на практике реализуется, так называемая, островная информатизация, что означает

- выделение в системе образования ключевых организационных, учебных, социальных и управленческих структур, допускающих интегральную информатизацию и способных служить «островами», начиная с которых может разворачиваться процесс глобальной информатизации образования;
- организацию проведения и обеспечения в этих подструктурах процесса системной интеграции информационных технологий;
- создание и поддержку условий, обеспечивающих по принципу цепной реакции распространение процесса разработки, развития и использования информационных технологий с «остро-

вов» информатизации на систему образования.»

В докладе также подчеркивается, что важнейшим условием информатизации образования является создание современной информационной среды, обеспечение доступа для системы образования России к современным информационным супермагистральям, к международным базам данных в области образования.

Примером того, какой может быть региональная информационная система управления в образовании при наличии достаточных ресурсов и развитой информационной среды, служит административная компьютерная система образовательного округа Jefferson County Public School в штате Кентукки, США. Указанный округ схож по количеству учащихся и территории (375 кв. миль) с небольшим российским регионом. Система обслуживает учреждения общего образования (школы, органы управления) и 7 региональных университетов, обеспечивает службу администрации округа информационными ресурсами и непосредственно поддерживает образовательный процесс. Ее основные функции:

- разгрузить учителей и администраторов от рутинной бумажной работы и освободить им время для работы с учащимися;
- предсказывать будущие потребности в ресурсах, позволяя управлению образованием округа быть активным и принимать опережающие решения;
- обеспечивать абсолютно все ресурсы, данные по грантам, региональным и федеральным программам, связанным с образованием, учащимися и школьным окружением.

Деятельность системы поддерживается региональной сетью, интегрирующей в себе большой центральный сервер на основе компьютера DPS8000 (класса main frame, с возможностью параллельно реагировать в диалоговом режиме реального времени на сотни запросов), пять мини-компьютеров BULL, поддерживающих коммуникации с центральным сервером, несколько тысяч персональных компьютеров и терминалов в школах и районных органах образования. Сеть работает под управлением ОС UNIX; она способна поддерживать передачу видео, звуковых, графических и текстовых данных.

Архитектурно сеть представляет собой звездообразную конструкцию с 14 подузлами, к которым подключены абоненты (порядка 1800 терминалов и 4000 телефонов по данным на 1996 г.). Абоненты подключены к подузлам низкоскоростными линиями связи на 9,6 кбайт/с, а подузлы связаны с центральным узлом микроволновыми линиями (через радиомодемы) со скоростью передачи данных 56 кбайт/с (после 1996 г. указанные скорости, скорее всего, как это планировалось, существенно увеличены).

Пользователи системы находятся более чем в 150 зданиях школ и административных центров. В число пользователей входят не только учебные заведения, но и родители, различные фирмы. На 1996 г. число пользователей равнялось примерно 3500 (учреждений и отдельных лиц). За 1995 г. система обслужила 250 000 транзакций.

Одна из основных функций системы - сбор данных обо всем, что связано с образованием (прежде всего в округе, но не только). Процесс сбора данных децентрализован. Пользователи вводят или актуализируют данные непосредственно с рабочих мест в школах или административных офисах. Собранные данные становятся немедленно доступными сообществу пользователей с соблюдением разумных ограничений по конфиденциальности и уровню; ограничения регулируются системой паролей пользователей при доступе к центральной базе данных.

Есть группа данных (и весьма обширных как по перечню, так и по объему), которые учебные заведения обязаны предоставлять в базу с установленной регулярностью (некоторые данные - практически ежедневно). Для этого в школах округа есть специальные должностные лица. К этим данным относятся

- демографическая информация по учащимся;
- результаты обучения;
- здоровье учащихся;
- школьный транспорт (в США доставка детей в школы и домой обязательна);
- квалификация учителей и другие.

В самом компьютерном центре функционирует служба централизованного сканирования данных. Она вводит в систему представляющие интерес данные, пришедшие иным, не электронным, путем. Эта же служба готовит весьма объемистые общие отчеты для управления образованием округа (раз в 6 недель), готовит материалы для централизованного тестирования учащихся, вы-

деляет пароли новым пользователям и делает другую необходимую работу. Служба готова в любой момент предоставить пользователям упорядоченные данные по сотням стандартных форм (скажем, по обучению взрослых - 107 форм, по посещаемости школ - 77 форм).

Весьма существенна для пользователей реализованная в обсуждаемой системе концепция интеграции данных. Данные доступны пользователю независимо от того, в какой форме и с помощью какого программного обеспечения они готовились. Данные также интегрированы по отношению к разнородным компьютерам сети. Пользователю безразлично, с какой машины и в каком формате к нему пришли запрашиваемые данные, они должны быть доступны ему по запросу без дальнейших усилий по перекодированию и т.п. Система первоначально не обладала таким качеством и это резко снижало ее практическую полезность. Все сказанное позволяет понять, почему общая стоимость используемого системой программного обеспечения оценивается в 90 млн. долларов.

Система развивается в следующих направлениях:

- движение от main frame к распределенным серверам;
- поддержка транзакций с большим объемом передаваемых данных;
- опережающее развитие среднего звена системы, базирующегося на UNIX, поддержка на этом уровне специальных транзакций (библиотечных, предпринимательских, служб социального сервиса и т.д.).

3.3. АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ НАУЧНЫХ ИССЛЕДОВАНИЙ

Автоматизированные системы для научных исследований (АСНИ) представляют собой программно-аппаратные комплексы, обрабатывающие данные, поступающие от различного рода экспериментальных установок и измерительных приборов, и на основе их анализа облегчающие обнаружение новых эффектов и закономерностей, рис. 6.9.

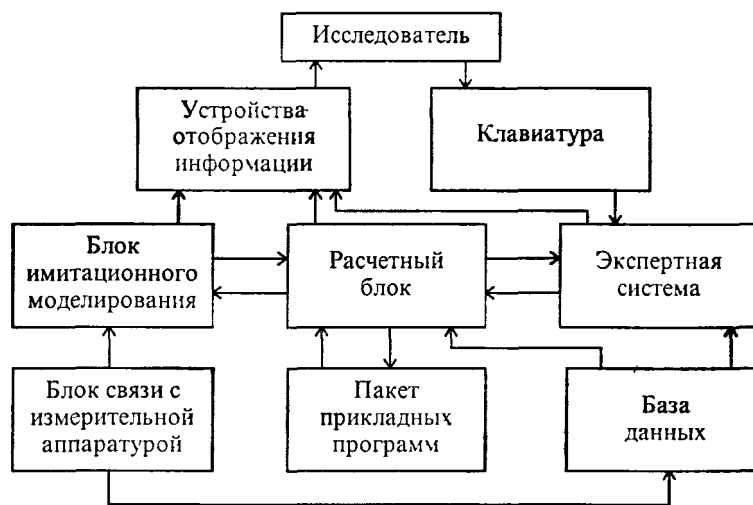


Рис. 6.9. Типовая структура АСНИ

Блок связи с измерительной аппаратурой преобразует к нужному виду информацию, поступающую от измерительной аппаратуры. В базе данных хранится информация, поступившая из блока связи с измерительной аппаратурой, а также заранее введенная с целью обеспечения работоспособности системы. Расчетный блок, выполняя программы из пакета прикладных программ, производит все математические расчеты, в которых может возникнуть потребность в ходе научных исследований. Расчеты могут выполняться по требованию самого исследователя, или блока имитационного моделирования. При этом на основе математических моделей воспроизводится процесс, происходящий во внешней среде.

Экспертная система моделирует рассуждения специалистов данной предметной области. С ее помощью исследователь может классифицировать наблюдаемые явления, диагностировать течение исследуемых процессов.

АСНИ получили широкое распространение в молекулярной химии, минералогии, биохимии, физике элементарных частиц и многих других науках.

3.4. СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

Близкими по своей структуре и функциям к системам автоматизации научных исследований оказываются системы автоматизированного проектирования (САПР), знакомство с которыми было начато в главе 2. Здесь мы ограничимся взглядом на САПР как на сложную информационную систему.

Проектирование новых изделий - основная задача изобретателей и конструкторов - протекает в несколько этапов, таких как формирование замысла, поиск физических принципов, обеспечивающих реализацию замыслов и требуемые значения параметров конструкции, поиск конструктивных решений, их расчет и обоснование, создание опытного образца, разработка технологии промышленного изготовления. Если формирование замысла и поиск физических принципов пока остаются чисто творческими, не поддающимися автоматизации этапами, то при конструировании и расчетах с успехом могут быть применены САПР, рис. 6.10.

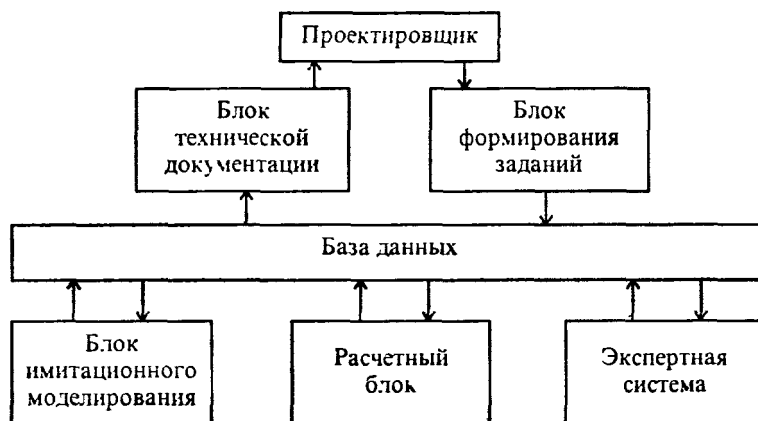


Рис. 6.10. Типовая схема САПР

База данных, блок имитационного моделирования, расчетный блок и экспертная система выполняют функции, аналогичные функциям соответствующих блоков АСНИ. Вместо блока связи с измерительной аппаратурой в САПР имеется блок формирования заданий. Проектировщик вводит в блок техническое задание на проектирование, в котором указаны цели, которые необходимо достичь при проектировании, и все ограничения, которые нельзя нарушить. Блок подготовки технической документации облегчает создание технической документации для последующего изготовления изделия.

В настоящее время САПР является неотъемлемым атрибутом крупных конструкторских бюро и проектных организаций, работающих в различных предметных областях. Это важная сфера приложения идей и методов информатики.

3.5. ГЕОИНФОРМАЦИОННЫЕ СИСТЕМЫ

Когда о человеке говорят «он прочно стоит на земле», то имеют в виду не только прямой смысл этих слов, но и нечто основательное в характере, положении, профессиональной квалификации. В информации, окружающей нас, тоже удивительно много «стоит на земле», хотя мы не всегда об этом задумываемся. Огромное количество практически необходимых знаний просто «подвисают», не будучи привязанными к тому участку земли, информацию о котором они несут. Те информационные системы, которые хранят эти знания, позволяют их актуализировать, сопоставлять, использовать для решения прикладных задач, называются географическими информационными системами, короче-геоинформационными системами (ГИС).

Как и многие виды информационных систем, ГИС уходят корнями в 60-е годы нашего века. Однако, их истинный расцвет состоялся лишь тогда, когда появились адекватные технические средства - огромные по емкости и скорости доступа носители информации и высококачественные графические визуальные устройства отображения информации - ведь в ГИС почти все разыгрывается на фоне географической карты. В нашей стране ГИС еще только «становятся на ноги»; специалисты предкают им большое будущее.

Приведем пример возможной ГИС. Муниципальная ГИС большого города (прообразы таковых имеются и в России) обслуживает всех тех, для кого информация привязана к месту ее нахождения в городе: городские власти, архитектурное управление, транспортников, предприятия по обслуживанию городских коммунальных сетей, энергетиков, связистов, работников торговли (магазинов, торговых баз), милицию, медицинские службы (особенно скорой помощи), налоговые службы, строительные организации, санитарно-эпидемиологические службы, органы социальной защиты и т.д. - всех трудно перечислить, ибо современный город является сложным социальным, экономическим и техническим образованием, и количество служб и учреждений, поддерживающих его жизнедеятельность, велико. Почти всегда нужная им информация привязана к карте города - как проехать скорой помощи, где произошел выброс вредных веществ и куда они распространяются, где перекрыть трубопровод при аварии, в каком состоянии транспортные магистрали и как проехать, если некоторые из них временно перекрыты, и прочее, и прочее. Потенциальный клиент такой ГИС - как любая городская служба, так и рядовые граждане, которые используют ее как информационно-справочную систему.

Для развертывания такой ГИС необходимо решить ряд сложных и дорогостоящих организационных задач, в том числе

- создания и ведения регулярно обновляемой цифровой (компьютерной) топографической основы;
- организации согласованного обновления пространственной информации, собираемой различными ведомствами;
- создания общегородских классификаторов основных структурных единиц города (улиц, микрорайонов и т.д.);
- создания единого координационного центра для ведения муниципальной ГИС.

По-видимому, муниципальный уровень является самым низким, на котором возможно создание многоцелевой ГИС того типа, который описан выше. На уровне региона или государства в целом информация столь велика по объему и столь тематически многообразна, что целесообразно создание тематических ГИС. Одной из важнейших задач, для решения которых такие региональные и общегосударственные ГИС должны быть вскоре разработаны, является создание государственного земельного кадастра, производимое на основе постановления правительства по федеральной целевой программе, принятой в августе 1996 г. Слово «кадастр» означает реестр, содержащий сведения об объекте. Цель разработки земельного кадастра - способствовать проведению единой политики в области земельных отношений, обеспечению интересов государства и населения страны, созданию цивилизованного рынка земли, защиты прав ее владельцев, арендаторов и т.д. Информация эта колоссальна по объему и требует постоянной актуализации, фиксирующей все изменения в сфере землепользования.

Создание ГИС может быть и объектом международного сотрудничества. Так, в период 1993 - 96 гг. усилиями шести стран создана ГИС «Черное море». Будучи жизненно важным для нескольких стран, море претерпевает катастрофические изменения, приводящие к сокращению и гибели целых экосистем. ГИС «Черное море» включает огромный объем картографической информации (более 2000 карт), базы данных по геологии, метеорологии, физической океанографии, загрязнениям, биоресурсам, рыбным ресурсам. Таким образом, государственные органы прибрежных стран, научные работники, да и просто все заинтересованные в судьбе моря получили возможность доступа к комплексу информации о нем.

Не следует думать, что каждая ГИС является столь огромной. Достаточно широкое распространение получили, так называемые, настольные ГИС. Они также хранят картографическую информацию и базы данных, привязанные к ней, но в гораздо более локальных вариантах. Скажем, для автовладельца большой интерес может представить ГИС, содержащая сведения о дорогах в районе, их покрытиях (асфальтовые, грунтовые и т.д.).

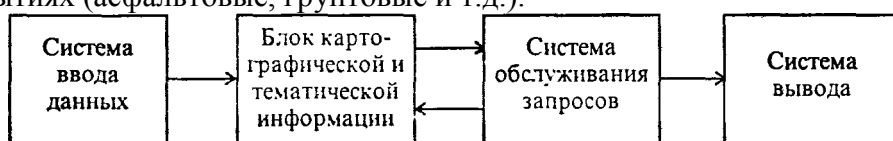


Рис. 6.11. Типовая структура ГИС

Для создания ГИС используют специализированные инструментальные программные средства, различные для разных классов ГИС. «Тяжелые» профессиональные системы типа Intergraph

не предназначены для персональных компьютеров (хотя и существуют их усеченные версии). Для создания локальных ГИС на ПК существуют специальные программные средства, работающие в среде MS Windows. Так, отечественные программы GeoDraw и GeoGraph 1.5 позволяют создать ГИС на основе многослойной топологической модели географических данных. Такая модель позволяет описать не только координаты объектов, но и их качественные характеристики (например, взаимное расположение), что важно при преобразованиях изображений. К каждому слою изображения может быть подключено несколько таблиц баз данных; наоборот, каждая таблица может быть подключена к нескольким слоям. Пользователь этой инструментальной системы может наполнить ее конкретным содержанием, рис. 6,11.

Особой проблемой в ГИС является ввод графической (особенно картографической) информации и выбор ее форматов. Если ввод карты может быть осуществлен сканированием, то, в отличие от многих других задач хранения, обработки и вывода изображений, растровый формат изображения, создаваемый при сканировании, в ГИС менее удобен, чем векторный. Дело в том, что графическая информация в ГИС часто подвергается манипуляциям типа «растянуть», «сжать» и более сложным геометрическим преобразованиям. Поэтому первоначальное растровое изображение в ГИС-системах обычно подвергается векторизации, т.е. установлению геометрических и формульных соотношений между линиями и точками, образующими изображение.

Контрольные вопросы

1. Каково назначение и основные структурные элементы АСУ?
2. В чем состоят недостатки концепции АСУ 70-х годов?
3. В чем состоят основные положения современной концепции информационной системы управления?
4. Каковы перспективы развития информационно-управляющих систем в образовании?
5. Каково назначение и основные структурные элементы АСНИ?
6. Каково назначение и основные структурные элементы САПР?
7. Каково назначение и основные структурные элементы ГИС?

§4. ЭКСПЕРТНЫЕ СИСТЕМЫ

Систему искусственного интеллекта, построенную на основе высококачественных специальных знаний о некоторой предметной области (полученных от экспертов - специалистов этой области), называют экспертной системой. Экспертные системы - один из немногих видов систем искусственного интеллекта (см. гл. 1) - получили широкое распространение и нашли практическое применение. Существуют экспертные системы по военному делу, геологии, инженерному делу, информатике, космической технике, математике, медицине, метеорологии, промышленности, сельскому хозяйству, управлению, физике, химии, электронике, юриспруденции и т.д. И только то, что экспертные системы остаются весьма сложными, дорогими, а главное, узкоспециализированными программами, сдерживает их еще более широкое распространение.

От других программ экспертные системы отличаются по следующим признакам:

- 1) компетентность - в конкретной предметной области экспертная система должна достигать того же уровня, что и эксперты - люди, при этом она должна пользоваться теми же эвристическими приемами, также глубоко и широко отражать предметную область;
- 2) символичные рассуждения - знания, на которых основана экспертная система, представляют в символическом виде понятия реального мира, рассуждения также происходят в виде преобразований символических наборов;
- 3) глубина - экспертиза должна решать глубокие, нетривиальные задачи, отличающиеся сложностью либо в плане сложности знаний, которые экспертная система использует, либо в плане их обилия, это не позволяет использовать полный перебор вариантов как метод решения задачи и заставляет прибегать к эвристическим, творческим, неформальным методам;
- 4) самосознание - экспертная система должна включать в себя механизм объяснения того, каким образом она приходит к решению задачи.

Экспертные системы (рис. 6.12) создаются для решения разного рода проблем, но основные типы их деятельности можно сгруппировать в категории, приведенные в табл. 6.6.

Экспертные системы, выполняющие интерпретацию, как правило, используют информа-

цию от датчиков для описания ситуации. Например, это может быть интерпретация показаний измерительных приборов на химическом заводе для определения состояния процесса. Интерпретирующие системы имеют дело не с четкими символьными представлениями проблемной ситуации, а непосредственно с реальными данными. Они сталкиваются с затруднениями, которых нет у систем других типов, потому что им приходится обрабатывать информацию зашумленную, недостаточную, неполную, ненадежную или ошибочную. Им необходимы специальные методы регистрации характеристик непрерывных потоков данных, сигналов или изображений и методы их символьного представления.

Интерпретирующие экспертные системы могут обрабатывать разнообразные виды данных. Например, системы анализа сцен и распознавания речи, используя

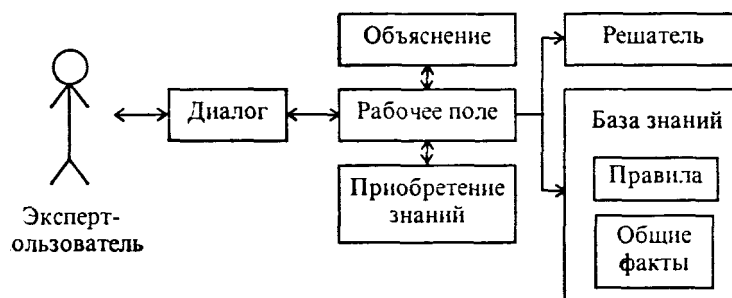


Рис. 6.12. Схема обобщенной экспертной системы

Таблица 6.6
Типичные категории способов применения экспертных систем

Категория	Решаемая проблема
Интерпретация	Описание ситуации по информации, поступающей от датчиков
Прогноз	Определение вероятных последствий заданных ситуаций
Диагностика	Выявление причин неправильного функционирования системы по наблюдениям
Проектирование	Построение конфигурации объектов при данных ограничениях
Планирование	Определение последовательности действий
Наблюдение	Сравнение результатов действий с ожидаемыми результатами
Отладка	Составление рецептов исправления неправильного функционирования системы
Ремонт	Выполнение последовательности предписанных исправлений
Обучение	Диагностика и исправление поведения обучаемого
Управление	Управление поведением системы как целого

естественную информацию (в одном случае визуальные образы, в другом - звуковые сигналы), анализируют их характеристики и понимают их смысл. *Интерпретация* в области химии использует данные дифракции рентгеновских лучей, спектрального анализа или ядерного магнитного резонанса для вывода химической структуры веществ. Интерпретирующая система в геологии использует каротажное зондирование - измерение проводимости горных пород в буровых скважинах и вокруг них, чтобы определить подповерхностные геологические структуры. Медицинские интерпретирующие системы используют показания следящих систем (например, значения температуры, пульса, кровяного давления), чтобы установить диагноз или тяжесть заболевания. Наконец, в военном деле интерпретирующие системы используют данные от радаров, радиосвязи и сонарных устройств, чтобы оценить ситуацию и идентифицировать цели.

Экспертные системы, осуществляющие *прогноз*, определяют вероятные последствия заданных ситуаций. Примерами служат прогноз ущерба урожаю от некоторого вида вредных насекомых, оценивание спроса на нефть на мировом рынке, прогнозирование места возникновения следующего вооруженного конфликта на основании данных разведки. Системы прогнозирования иногда используют имитационное моделирование, т.е. программы, которые отражают причинно-следственные взаимосвязи в реальном мире, чтобы сгенерировать ситуации или сценарии, которые могут возникнуть при тех или иных входных данных (конкретный пример приведен в главе 7).

Эти возможные ситуации вместе со знаниями о процессах, порождающих эти ситуации, образуют предпосылки для прогноза. Специалисты по искусственному интеллекту пока что разработали сравнительно мало прогнозирующих систем, возможно потому, что очень трудно взаимодействовать с имитационными моделями и создавать их.

Экспертные системы выполняют *диагностирование*, используя описания ситуаций, характеристики поведения или знания о конструкции компонентов, чтобы установить вероятные причины неправильно функционирования диагностируемой системы. Примерами служат определение причин заболевания по симптомам, наблюдаемым у пациентов; локализация неисправностей в электронных схемах и определение неисправных компонентов в системе охлаждения ядерных реакторов. Диагностические системы часто являются консультантами, которые не только ставят диагноз, но и помогают в отладке. Они могут взаимодействовать с пользователем, чтобы оказать помощь при поиске неисправностей, а затем предложить порядок действий по их устранению. Медицина представляется вполне естественной областью для диагностирования, и действительно, в медицинской области было разработано больше диагностических систем, чем в любой другой отдельно взятой предметной области. Однако в настоящее время многие диагностические системы разрабатываются для приложений к инженерному делу и компьютерным системам.

Экспертные системы, выполняющие *проектирование*, разрабатывают конфигурации объектов с учетом набора ограничений, присущих проблеме. Примерами могут служить генная инженерия, разработка СБИС и синтез сложных органических молекул.

Экспертные системы, занятые *планированием*, проектируют действия; они определяют полную последовательность действий, прежде чем начнется их выполнение. Примерами могут служить создание плана применения последовательности химических реакций к группам атомов с целью синтеза сложных органических соединений или создание плана воздушного нападения, рассчитанного на несколько дней, с целью нейтрализации определенного фактора боеспособности врага.

Экспертные системы, выполняющие *наблюдение*, сравнивают действительное поведение с ожидаемым поведением системы. Примерами могут служить слежение за показаниями измерительных приборов в ядерных реакторах с целью обнаружения аварийных ситуаций или оценка данных мониторинга больных, помещенных в блоки интенсивной терапии. Наблюдающие экспертные системы подыскивают наблюдаемое поведение, которое подтверждает их ожидания относительно нормального поведения или их предположения о возможных отклонениях. Наблюдающие экспертные системы по самой своей природе должны работать в режиме реального времени и осуществлять зависящую как от времени, так и от контекста интерпретацию поведения наблюдаемого объекта.

Экспертные системы, выполняющие *обучение*, подвергают диагностике, «отладке» и исправлению (коррекции) поведение обучаемого. В качестве примеров приведем обучение студентов отысканию неисправностей в электрических цепях, обучение военных моряков обращению с двигателем на корабле и обучение студентов-медиков выбору антимикробной терапии. Обучающие системы создают модель того, что обучающийся знает и как он эти знания применяет к решению проблемы. Системы диагностируют и указывают обучающемуся его ошибки, анализируя модель и строя планы исправлений указанных ошибок. Они исправляют поведение обучающихся, выполняя эти планы с помощью непосредственных указаний обучающимся.

Экспертные системы, осуществляющие *управление*, адаптивно руководят поведением системы в целом. Примером служит управление производством и распределением компьютерных систем. Управляющие экспертные системы должны включать наблюдающие компоненты, чтобы отслеживать поведение объекта на протяжении времени, но они могут нуждаться и в других компонентах для выполнения любых или всех из уже рассмотренных типов задач: интерпретации, прогнозирования, диагностики, проектирования, планирования, отладки, ремонта и обучения. Типичная комбинация задач состоит из наблюдения, диагностики, отладки, планирования и прогноза.

Рассмотрим примеры наиболее известных, классических экспертных систем, с которых началось создание и развитие этого типа программных средств.

MYCIN - это экспертная система, разработанная для медицинской диагностики. В частности, она предназначена для работы в области диагностики и лечения заражения крови и медицинских инфекции. Система ставит соответствующий диагноз, исходя из представленных ей симпто-

мов, и рекомендует курс медикаментозного лечения любой из диагностированных инфекций. Она состоит в общей сложности из 450 правил, разработанных с помощью группы по инфекционным заболеваниям Стэнфордского университета. Ее основополагающим моментом является использование вероятностного подхода.

Система MYCIN справляется с задачей путем назначения показателя определенности каждому из своих 450 правил. Поэтому можно представлять MYCIN как систему, содержащую набор правил вида «ЕСЛИ... ТО» с определенностью P , которые предоставили люди - эксперты - и которые изложили и правила и указали свою степень доверия к каждому правилу по шкале от 1 до 10.

Установив эти правила и связанные с ними показатели определенности, MYCIN идет по цепочке назад от возможного исхода, чтобы убедиться, можно ли верить такому исходу. Установив все необходимые исходные предпосылки, MYCIN формирует суждение по данному исходу, рассчитанное на основе показателей определенности, связанных со всеми правилами, которые нужно использовать.

Допустим, чтобы получить исход Z , требуется определить предпосылки X и Y , дающие возможность вывести Z . Но правила для определения X и Y могут иметь связанные с ними показатели определенности P и Q . Если значения P и Q были равны 1,0, то исход Z не вызывает сомнения. Если P и Q меньше 1,0 (как это обычно бывает), то исход Z не последует наверняка. Он может получиться лишь с некоторой степенью определенности.

MYCIN не выдает диагноз и не раскрывает его точный показатель неопределенности. Система выдает целый список диагнозов, называя показатель определенности для каждого из них. Все диагнозы с показателями выше определенного специфического для каждого диагноза уровня принимаются как в той или иной степени вероятные, и пользователю вручается список возможных исходов.

Стандартные фразы и грамматические формы были без труда приспособлены к программе, и в результате получился существенно вырожденный диалект английского языка, легко поддающийся программированию.

Врачи оказались очень довольными таким результатом, потому что, сами не осознавая этого, они говорили, используя очень небольшой набор слов английского языка (по крайней мере, когда сообщали о своей работе).

В некотором роде это имеет нечто общее с системой DENDRAL, в которой применяется графический язык, приспособленный к специфической деятельности химиков.

DENDRAL - это старейшая, самая разработанная экспертная система в мире. Или, по крайней мере, старейшая система, названная экспертной.

Химик, получив какое-то вещество, часто хочет знать, какова его химическая структура. Для этого существуют различные способы. Во-первых, специалист может сделать определенные умозаключения на основе собственного опыта. Во-вторых, он может исследовать это вещество на спектрометре и, изучая структуру спектральных линий, уточнить свои первоначальные догадки. Во многих случаях это даст ему возможность точно определить структуру вещества.

Проблема состоит в том, что все это требует времени и значительной экспертизы со стороны научного сообщества. Здесь и оказывается очень полезной система DENDRAL, автоматизирующая процесс определения химической структуры вещества.

В самых общих чертах процесс принятия решения следующий. Пользователь дает системе DENDRAL некоторую информацию о веществе, а также данные спектрометрии (инфракрасной, ядерного магнитного резонанса и масс-спектрометрии), и та, в свою очередь, выдает диагноз в виде соответствующей химической структуры.

Можно для простоты представить систему DENDRAL состоящей из двух частей, как если бы в одной экспертной системе были две самостоятельные системы. Первая из них содержит набор правил для выработки возможных химических структур. Вводимая информация состоит из ряда заключений, сделанных химиком, и позволяет судить, какие структуры вероятны в том или ином случае.

На выходе первой системы имеется не один простой ответ. Обычно это целая серия возможных структур - программа не в состоянии точно сказать, какая из них верна. Затем DENDRAL «берет» каждую из этих структур по очереди и использует вторую экспертную систему, чтобы определить для каждой из них, каковы были бы результаты спектрального анализа, если бы это вещество существовало и было на самом деле исследовано по спектрограмме.

Процесс, часто именуемый «генерация и проверка», позволяет постоянно сокращать число возможных рассматриваемых вариантов, чтобы в любой момент оно было как можно меньше. В отличие от некоторых экспертных систем DENDRAL задумана не как «игрушка». Она не используется лишь для проверки теоретических основ экспертных систем, а реально применяется для определения химических структур.

PROSPECTOR - это экспертная система, применяемая при поиске коммерчески оправданных месторождений полезных ископаемых.

Система PROSPECTOR, по аналогии с MYCIN, содержит большое число правил, относящихся к различным объектам, а также возможных исходов, выведенных на их основе. В этой системе используется также «движение по цепочке назад» и вероятности. Методы этой системы являются одними из лучших среди всех разработанных методов для любой из существующих ныне систем.

Самый простой случай - правила, выражающие логические отношения. Это правила типа «ЕСЛИ x , ТО z », где событие « z » непосредственно вытекает из « x ». Они остаются столь же простыми, если сопоставить « x » вероятность.

Если у « x » всего один аргумент, то это правило существенно упрощается. Обычно вместо « x » мы представляем более сложное логическое выражение, например (x И y) или (x ИЛИ y).

Если элементы отношения связаны с помощью логического И и отдельным элементам этого отношения сопоставлены определенные вероятности, то система PROSPECTOR выбирает минимальное из этих значений и присваивает эту минимальную вероятность рассматриваемому возможному исходу. Поэтому, когда вероятность $x = 0,1$, вероятность $y = 0,2$, вероятность исхода $z = 0,1$. Легко видеть, почему выбран такой метод: чтобы « z » было истинным, и « x », и « y » должны быть истинными. Это является «жестким» ограничением, поэтому следует брать минимальное значение.

Система PROSPECTOR пользуется методом, основанным на применении формулы Байеса с целью оценки априорной и апостериорной вероятностей какого-либо события.

В целом правила в системе PROSPECTOR записываются в виде ЕСЛИ..., ТО (LS, LN), причем каждое правило устанавливается с отношением правдоподобия как для положительного, так и для отрицательного ответа.

Система PROSPECTOR предлагает пользователю шкалу ответов в диапазоне от -5 до +5. Нижний предел - это определенно «Да», верхний - определенно «Нет».

Обычно ответ пользователя находится где-то между крайними значениями, и PROSPECTOR корректирует $P(H)$, учитывая LS и LN с помощью линейной интерполяции. Это легко представить себе в виде линейной шкалы, где LN - крайнее левое, а LS - крайнее правое значение.

Кроме экспертных систем MYCIN, DENDRAL и PROSPECTOR существует большое количество других экспертных систем. Ниже (табл. 6.7) приводится список некоторых систем, отличительной особенностью которых является наличие большой базы знаний. Этот перечень, конечно, весьма неполный, потому что в последнее время происходит быстрое расширение сферы применения экспертных систем, и полный их перечень был бы огромным и устарел бы почти сразу же после его опубликования.

В этом списке приведены также «пустые» экспертные системы (не содержащие конкретных правил предметных областей) и экспертные системы по построению других экспертных систем. Такие системы являются инструментальными средствами для создания прикладных экспертных систем. Они значительно облегчают задачи создания полномасштабной прикладной экспертной системы.

Вообще же инструментальные средства создания экспертных систем (ЭС) классифицируют следующим образом:

- символьные языки программирования, ориентированные на создание ЭС и систем искусственного интеллекта (например, LISP, INTERLISP, SMALLTALK);
- языки инженерии знаний, т.е. языки высокого уровня, ориентированные на построение ЭС (например, OPS-5, LOOPS, Пролог, KES);
- системы, автоматизирующие разработку (проектирование) ЭС (например, KEE, ART, TEIRESIAS, AGE, TIMM); их часто называют окружением (environment) для разработки систем искусственного интеллекта, ориентированных на знания;
- оболочки ЭС (или пустые ЭС) - ЭС, не содержащие знания ни о какой проблемной области

(например, ЭКСПЕРТИЗА, EMYCIN, ЭКО, ЭКСПЕРТ).

Таблица
Список некоторых экспертных систем

Наименование системы	Назначение системы
MYCIN, PUFF, PIP, CASNET, INTERNIST SACON PROSPECTOR DENDRAL SECHS SYNCHEM EL MOLGEN MECHO PECOS RI SU/X VM SOPHIE GUIDON TEIRESIAS, EMYCIN, EXPERT, KAS ROSIE, AGE, HEARSAY III, AL/X, SAGE, Micro-Expert	Медицинская диагностика Техническая диагностика Геологическая диагностика Определение химической структуры вещества Определение химической структуры вещества Анализ цепей Генетика Механика Программирование Конфигурирование компьютеров Машинная акустика Медицинские измерения Обучение электронике Медицинское обучение Построение базы данных Построение экспертных систем

Контрольные вопросы и задания

1. Что отличает экспертные системы от других программ?
2. Какие категории различных типичных проблем, решаемых экспертными системами, можно выделить?
3. Охарактеризуйте экспертную систему MYCIN.
4. Охарактеризуйте экспертную систему DENDRAL.
5. Охарактеризуйте экспертную систему PROSPECTOR.
6. Какие виды инструментальных программных средств для создания экспертных систем существуют?
7. Какие еще экспертные системы вы знаете?

§ 5. КОМПЬЮТЕРНЫЕ ОБУЧАЮЩИЕ СИСТЕМЫ

5.1. ОСНОВНЫЕ ПРИНЦИПЫ НОВЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ОБУЧЕНИЯ

Как уже неоднократно отмечалось, создание и совершенствование компьютеров привело и продолжает приводить к созданию новых технологий в различных сферах научной и практической деятельности. Одной из таких сфер стало образование - процесс передачи систематизированных знаний, навыков и умений от одного поколения к другому. Будучи само по себе мощной информационной сферой и владея опытом использования различных классических (не компьютерных) информационных систем, образование быстро откликнулось на возможности современной техники. На наших глазах возникают нетрадиционные информационные системы, связанные с обучением; такие системы естественно называть информационно-обучающими.

С началом промышленного изготовления компьютеров первых поколений и их появлением в образовательных учреждениях возникло новое направление в педагогике - компьютерные техно-

логии обучения. Первая обучающая система Plato на основе мощной ЭВМ фирмы «Control Data Corporation» была разработана в США в конце 50-х годов и развивалась в течение 20 лет. По-настоящему массовыми создание и использование обучающих программ стали с начала 80-х годов, когда появились и получили широкое распространение персональные компьютеры. С тех пор образовательные применения ЭВМ выдвинулись в число их основных применений наряду с обработкой текстов и графики, оттеснив на второй план математические расчеты.

С появлением примеров компьютерного обучения к созданию компьютерных обучающих программ приобщились десятки тысяч педагогов - специалистов в различных областях знания, чаще всего в технических науках. В разрабатываемых ими программах, опираясь в основном на интуицию и практический опыт, они воплощали свои представления о преподавании конкретных дисциплин с помощью компьютеров. Педагоги-теоретики долгое время оставались в стороне от этого нового направления в обучении. В результате до сих пор отсутствует общепризнанная психолого-педагогическая теория компьютерного обучения, компьютерные обучающие программы продолжают создаваться и применяться без необходимого учета принципов и закономерностей обучения.

Благодаря своим конструктивным и функциональным особенностям современный персональный компьютер является уникальной по своим возможностям обучающей машиной. Он находит применение в обучении самым разнообразным дисциплинам и служит базой для создания большого числа новых информационных технологий обучения. Какие же особенности персонального компьютера так выгодно отличают его от прежде известных обучающих машин и технических средств обучения?

Это не столько какая-то одна возможность персонального компьютера, сколько сочетание

- интерактивного (диалогового) режима работы (действие человека - реакция компьютера - ... - действие человека - реакция компьютера и т.д.);

- «персональности» (небольшие размеры и стоимость, позволяющие обеспечить компьютерами целый класс);

- хороших графических, иллюстративных возможностей (экраны распространенных модификаций имеют разрешающую способность 640x480 точек при 16 млн цветовых оттенков - это качество хорошего цветного телевизора или журнальной иллюстрации);

- простоты управления, наличия гибких языков программирования человеко-машинного диалога и компьютерной графики;

- легкости регистрации и хранения информации о процессе обучения и работе учащегося, а также возможности копирования и размножения обучающих программ.

Технические возможности персонального компьютера, если компьютер используется как обучающее средство, позволяют

- активизировать учебный процесс;
- индивидуализировать обучение;
- повысить наглядность в предъявлении материала;
- сместить акценты от теоретических знаний к практическим;
- повысить интерес учеников к обучению.

Активизация обучения связана с диалоговым характером работы компьютера и с тем, что каждый ученик работает за своим компьютером. При традиционном классном обучении основное - это восприятие учащимися информации в устной форме, при этом ученику не часто приходится проявлять активность на уроке и учитель не в состоянии организовать и контролировать активную работу каждого ученика на его рабочем месте. Поэтому традиционное обучение, в основном, является пассивным - многие педагоги сетуют, что на уроке активно работают 20 -30% учащихся. Если же обучение ведется в компьютерном классе, компьютер диалоговым характером своей работы стимулирует ученика к деятельности и контролирует ее результаты.

Индивидуализация обучения при использовании компьютера также связана с интерактивным характером работы с компьютером и наличием компьютеров на рабочих местах: каждый ученик теперь может сам выбирать темп обучения, делать в работе паузы. Более глубокий и тонкий учет индивидуальных особенностей учащихся может осуществлять компьютерная программа, с помощью которой ведется обучение (педагогическое программное средство, сокращенно ППС). С помощью начального теста программа может определить уровень обученности ученика, и в соответствии с этим уровнем предъявлять теоретический материал, вопросы и задачи, а также под-

сказки и помощь. Обучение слабых учеников программа ведет на самом легком (базовом) уровне, изложение теоретических сведений максимально упрощено, вопросы и задачи облегчены, помощь имеет характер прямой подсказки. Обучение сильных учеников ведется на наиболее сложном уровне, теория излагается углубленно, предлагаются творческие задачи, требующие изобретательности и интуиции, а помощь имеет косвенный характер - намек или наводящего на правильный путь соображения. Между этими крайними случаями обучающая программа может учитывать более тонкую градацию подготовленности учащихся.

Каждый ученик в процессе обучения сталкивается с трудностями индивидуального характера, связанными с наличием пробелов в знаниях или особенностями мышления. При обучении с помощью компьютера обучающая программа может диагностировать пробелы в знаниях ученика, его индивидуальные особенности и строить обучение в соответствии с ними.

Графические возможности дисплеев персональных компьютеров и гибкие языки программирования позволяют сделать компьютерное обучение очень наглядным. В самом деле, теперь на каждом рабочем месте ученика имеется телевизор - дисплей, на экране которого с помощью языка программирования можно без всякой кино- и видеосъемки показывать геометрические фигуры и построения, стилизованные изображения реальных объектов и т.п. - и все это как статически (т.е. неподвижно), так и динамически, в движении. С помощью компьютерной графики можно сделать зримыми или, как еще говорят, визуализировать такие явления и процессы, которые не могут быть увидены в действительности (тем более в условиях школьного класса), можно создать наглядный образ того, что на самом деле никакой наглядности не имеет (например, эффектов теории относительности, закономерностей числовых рядов и т.п.). На этой возможности компьютеров основывается, так называемая, когнитивная компьютерная графика - особое направление применения компьютеров в научных исследованиях, когда-иллюстративные возможности компьютера используются для изучения различных закономерностей.

Всегда остро стоит вопрос о соотношении теории и практики применительно к научному знанию, обучению и т.д. (на это обращал внимание еще гетевский Мефистофель: «Суша теория, мой друг, но древо жизни вечно зеленеет»). Традиционное обучение является преимущественно теоретическим. Классно-урочная форма обучения исподволь, незаметно подталкивает каждого педагога в отдельности и всю систему образования в целом к усилению теоретической стороны обучения в ущерб практической. В самом деле, любому педагогу излагать теоретические знания у доски и требовать от учеников воспроизведения этого изложения значительно легче, чем организовывать ориентированную на практику работу учащихся. Если же вести обучение с помощью компьютера, оно приобретает практический уклон: диалоговый характер работы с компьютером, его вычислительные моделирующие возможности предрасполагают к обучению в форме решения задач (и к тому же задач практической направленности).

Важным условием успешного обучения является интерес учеников и изучаемому предмету, ходу обучения и его результату. Этот интерес связан со множеством факторов: содержанием изучаемого предмета, уровнем его сложности, организацией процесса обучения, системой поощрений и наказаний, применяемой учителем, личностными качествами самого учителя (его мастерством и интересом к предмету), системой ценностей ученика, его ближайшего окружения, родителей, взаимоотношениями в классном коллективе, социальным заказом в подготовке по направлению науки, представляемому данным предметом. В последнее десятилетие действует очень настоятельный социальный заказ в отношении всего, что связано с компьютерами (в подготовке специалистов по компьютерам и их применению, в развитии компьютерных технологий, в распространении компьютерной грамотности -умению использовать компьютер для решения разнообразных прикладных задач в различных сферах профессиональной деятельности).

Действию скрытого социального заказа мы обязаны появлением большого числа «компьютерных» талантов и дарований. Сфера деятельности, связанная с компьютером, непосредственная работа на компьютере сама по себе обладает привлекательными чертами, втягивает в себя людей. Существует даже особая категория людей («хакеров»), увлекшихся сложными и тонкими вопросами управления компьютерами, программированием различных компьютерных эффектов. В некоторых случаях можно говорить даже о возникновении психологической зависимости человека от компьютера - настолько велико мотивирующее влияние компьютера.

Компьютерная технология повышает интерес к обучению предметам, не связанным с информатикой. Новое в организации учебного процесса с участием компьютера, само изменение ха-

рактера работы ученика на уроке способствуют повышению интереса к учебе. В то же время, более тонкое использование возможностей компьютера позволяет управлять мотивацией учеников во время компьютерного обучения. Здесь имеются в виду, в первую очередь, мотивирующие реплики обучающих программ, т.е. фразы, в которых обучающая программа оценивает работу ученика и стимулирует дальнейшее обучение. Эти фразы могут иметь неформальный характер с оттенком юмора и создавать теплую партнерскую эмоциональную атмосферу при работе с компьютером. Важное значение имеют элементы игры, состязательности в компьютерном обучении (например, подсчет очков и сравнение достижений различных учеников) или звуковые и зрительные эффекты (звучание музыкальных мелодий, мигание и цвета на экране дисплея).

Вот далеко неполный арсенал возможностей компьютера, делающих его очень перспективным для использования в учебном процессе обучающим средством.

Итак, компьютеры - эти уникальные по своим возможностям обучающие машины - установлены в классе... И тут выясняется, что не понятно, как к этим компьютерам подступиться, т.е. говорить о компьютерном обучении еще рано. Как быть, с чего начать переход к компьютерному обучению?

Ответ таков: «с подбора обучающих программ и продумывания организационных форм их применения, с разработки методик, использующих возможности компьютера в обучении». Нельзя рассматривать компьютер в обучении (да и в других сферах тоже) отдельно, сам по себе, в отрыве от

- а) программного обеспечения - педагогических программных средств;
- б) организационных форм использования компьютеров.

В настоящее время существует огромное множество обучающих программ по самым разным предметам, ориентированных на самые различные категории учащихся, начиная контингентом детских садов и кончая персоналом атомных электростанций. Кроме того, каждая из программ предназначена только для одного типа компьютеров - а ведь этих типов великое множество - и не годится для других! Далее будем иметь в виду лишь обучающие программы по общеобразовательным предметам средней школы. Их очень много, и четкая классификация разновидностей этих программ еще не установилась.

5.2. ТИПЫ ОБУЧАЮЩИХ ПРОГРАММ

Основанием для классификации служат обычно особенности учебной деятельности обучаемых при работе с программами. Многие авторы выделяют четыре типа обучающих программ:

- тренировочные и контролирующие;
- наставнические;
- имитационные и моделирующие;
- развивающие игры.

Программы 1-го типа (тренировочные) предназначены для закрепления умений и навыков. Предполагается, что теоретический материал уже изучен. Эти программы в случайной последовательности предлагают учащемуся вопросы и задачи и подсчитывают количество правильно и неправильно решенных задач (в случае правильного ответа может выдаваться поощряющая ученика реплика). При неправильном ответе ученик может получить помощь в виде подсказки.

Программы 2-го типа (наставнические) предлагают ученикам теоретический материал для изучения. Задачи и вопросы служат в этих программах для организации человеко-машинного диалога, для управления ходом обучения. Так если ответы, даваемые учеником, неверны, программа может «откатиться назад» для повторного изучения теоретического материала.

Программы наставнического типа являются прямыми наследниками средств программированного обучения 60-х годов в том смысле, что основным теоретическим источником современного компьютерного или автоматизированного обучения следует считать программированное обучение. В публикациях зарубежных специалистов и сегодня под термином «программированное обучение» понимают современные компьютерные технологии. Одним из основоположников концепции программированного обучения является американский психолог Б.Ф.Скиннер.

Главным элементом программированного обучения является программа, понимаемая как упорядоченная последовательность рекомендаций (задач), которые передаются с помощью дидактической машины или программированного учебника и выполняются обучаемыми. Существует

несколько известных разновидностей программированного обучения.

1. Линейное программированное обучение. Основатель - Б.Ф.Скиннер, профессор психологии Гарвардского университета, США. Впервые выступил со своей концепцией в 1954 г. При ее создании Скиннер опирался на бихевиористскую психологию, в соответствии с которой обучение основано на принципе S - R, т.е. на появлении некоторых факторов (S - stimulus) и реакции на них (R - reaction). По этой концепции для любой реакции, соответственно усиленной, характерна склонность к повторению и закреплению. Поощрением для обучаемого является подтверждение программой каждого удачного шага, причем, учитывая простоту реакции, возможность совершения ошибки сводится к минимуму.

Линейная программа в понимании Скиннера характеризуется следующими особенностями:

- дидактический материал делится на незначительные дозы, называемые шагами, которые обучаемые преодолевают относительно легко, шаг за шагом;
- вопросы, содержащиеся в отдельных рамках программы, не должны быть очень трудными, чтобы обучаемые не потеряли интереса к работе;
- обучаемые сами дают ответы на вопросы, привлекая для этого необходимую информацию;
- в ходе обучения учащихся сразу же информируют о том, правильны или ошибочны их ответы;
- все обучаемые проходят по очереди все рамки программы, но каждый делает это в удобном ему темпе;
- во избежание механического запоминания информации одна и та же мысль повторяется в различных вариантах и нескольких рамках программы.

2. Разветвленная программа. Автор концепции разветвленного программирования - Норман А.Кроудер. Разветвленная программа основана на выборе одного правильного ответа из нескольких данных, она ориентирует на текст многократного выбора. По мнению автора, выбор правильных ответов требует от обучаемых больших умственных способностей, нежели припоминание какой-то информации. Непосредственное подтверждение правильности ответа он считает своеобразным типом обратной связи.

Вопросы, в понимании Кроудера, имеют целью

- проверить, знает ли ученик материал;
- в случае отрицательного ответа отсылать обучаемого к координирующим и соответственно обосновывающим ответ порциям информации;
- возможность закрепления основной информации с помощью рациональных упражнений;
- увеличение усилий обучаемого и одновременную ликвидацию механического обучения через многократное повторение информации;
- формирование требуемой мотивации обучаемого.

Если основой линейной программы является стремление избежать ошибок, то разветвленная программа не направлена на ликвидацию ошибок в процессе обучения; ошибки Кроудер трактует как возможность обнаружить недостатки в знаниях обучаемых, а также выяснить, какие проблемы обучаемые уяснили недостаточно; благодаря этому о его программе можно было бы сказать, что она сводится к «управлению процессом мышления», в то время как линейная программа основана на «управлении ответами».

Постепенно оба классических типа - линейное и разветвленное программированное обучение-уступили место смешанным формам.

По своей методической структуре педагогическое программное средство (ППС), реализующие программированный подход, характеризуются наличием следующих блоков:

- блока ориентировочной основы действий (ООД), содержащего текстово-графическое изложение теоретических основ некоторого раздела автоматизированного курса;
- контрольно-диагностического блока, контролирующего усвоение ООД и управляющего обучением;
- блока автоматизированного контроля знаний, формирующего итоговую оценку знаний учащегося.

Известно несколько видов организации программ наставнического типа, называемых также алгоритмами программированного обучения.

1. Последовательно-подготовительный алгоритм. Начальный элемент задания относи-

тельно прост, он подготавливает выполнение второго, более сложного, а тот, в свою очередь, третьего и т.д. Заключительные элементы имеют достаточно высокий уровень сложности.

2. *Параллельно-подготовительный алгоритм.* Начальные элементы заданий независимо один от другого подготавливают выполнение следующего за ним комплексного элемента высокого уровня.

3. *Последовательно-корректирующий алгоритм.* Начальные элементы задания имеют высокий уровень сложности, а каждый последующий элемент корректирует выполнение предыдущего, указывая, например, на противоречия, к которым приводят неправильные ответы.

4. *Параллельно-корректирующий алгоритм.* Обучаемому предлагается комплексный элемент высокого уровня, последующие элементы играют роль наводящих (подсказывающих), причем с разных позиций, независимо один от другого.

5. *Алгоритм переноса.* Приводятся два массива элементов $A(N)$ и $B(N)$. Ими могут быть понятия, отношения, действия, характеристики и т.д. Требуется установить логическое соответствие между ними.

6. *Аналитический алгоритм.* Предлагаются элементы $A(N)$. Необходимо установить принадлежность каждого из них к одному из классов $B(K)$.

7. *Синтезирующий алгоритм.* Элементы массива $A(N)$ уже разбиты на подгруппы. Задача обучаемого - установить критерий, по которому осуществлялась классификация.

8. *Алгоритм упорядочения.* Элементы массива $A(N)$ необходимо упорядочить по некоторому указанному критерию $B(K)$. Этот алгоритм требует для своего выполнения комплексной умственной деятельности.

Большинство инструментальных систем предоставляют преподавателю возможность составлять обучающие и контролирующие задания с различными типами ответов

1. *С выборочным ответом.* Обучаемому предлагается задание (вопрос) и набор (меню) готовых ответов, из которых он может сделать выбор правильного, по его мнению, ответа (утверждения).

Такой вариант задания наиболее удобен для машинной реализации, так как ЭВМ анализирует лишь номер, по которому легко определяет правильность ответа. На первый взгляд задания с выборочным ответом имеют ряд недостатков, а именно: обязательное предъявление верного ответа, возможность его угадывания, а значит, ограничение мыслительной деятельности обучаемого. Эти недостатки существенно снижаются путем правильного, творческого и остроумного применения различных принципов составления таких заданий.

Вероятность угадывания правильного ответа сводится к минимуму следующими простыми приемами:

- повторением аналогичного по смыслу вопроса в нескольких различных формах;
- увеличением числа элементов для выбора (при выборе из пяти ответов вероятность угадывания равна 0,2):
- увеличение числа верных ответов до двух или до нескольких пар. Подбирать ответы в заданиях необходимо таким образом, чтобы они были правдоподобными и равнопривлекательными.

2. *С частично-конструируемым ответом.* Задания этого типа являются промежуточным и связующим звеном между заданиями с выборочным ответом и свободно-конструируемым. Частично-конструируемый ответ составляется из частей, предложенных преподавателем.

Эта форма используется для заданий по составлению определений законов, теорем, стандартных формулировок и т.д. В верный ответ входят, как правило, не все элементы задания, и порядок их выбора не является жестким.

3. *Со свободно-конструируемым ответом.* Задания такого типа являются наиболее предпочтительными для автоматизированного обучения и контроля. Они позволяют слушателю общаться с компьютером на естественном языке, имитируя диалог обучаемого и преподавателя. Задания со свободно-конструируемым ответом наиболее сложны для обучаемого, так как полностью исключают возможность угадывания и требуют значительной умственной работы перед вводом в компьютер ответа, набираемого на клавиатуре в свободной форме. В то же время резко возрастает сложность деятельности преподавателя - автора курса по формированию автономных ответов для анализатора инструментальной системы.

Эталон может содержать, как правило, не более 80 символов, включая пробелы. Ответ обучаемого на заданный вопрос сравнивается с текстом эталона и вырабатывается соответствующий

признак ответа: «верный», «неверный», «предполагаемый» и т.д. Далее программа переходит к тому кадру сценария, который соответствует полученному признаку.

Таким образом, автор курса формирует кадры, предъявляемые обучаемому в зависимости от признака ответа, что создает иллюзию «понимания» системой смысла введенной фразы, так как при разных ответах на один и тот же вопрос обучаемый получает и различную реакцию компьютера.

В современных инструментальных системах реализованы следующие методы сравнения эталонного ответа с ответом обучаемого.

1. *Анализ по ключевым словам.* Этот метод анализа достаточно прост и универсален. Эталонный ответ, заранее введенный преподавателем, используется в качестве ключа, который сравнивается с ответом обучаемого на протяжении всей строки. Ключом может быть один символ, слово или группа слов.

При использовании ключевых слов можно достичь достаточно хороших результатов. Но применять метод надо достаточно осторожно, так как возможности распознавания смысла с его помощью ограничены. Недостаток ключевого поиска выражается в том, что ответ не распознается при перестановках внутри ключа.

2. *Синтаксический анализ* с использованием символов частичной обработки ответа обучаемого. Этот метод анализа целесообразно использовать в том случае, когда требуется выполнить сравнение не по ключу, а по жесткому эталону. Лишний символ должен считаться ошибкой, пробелы не игнорируются. Выполняется как бы прямое (посимвольное) сравнение посимвольного ответа с эталоном. При совпадении всех символов ответа с символами эталона вырабатывается признак «верно».

Однако при работе обучаемых с курсом могут возникнуть ситуации, когда необходимо, с целью более корректного толкования смысла ответа, сделать некоторые отступления от правил прямого сравнения. В подобных ситуациях метод синтаксического анализа предусматривает средства частичной обработки ответов обучаемого.

Символы частичной обработки ответа (спецсимволы), включенные в эталон ответа, позволяют исключить, игнорировать в ответе обучаемого один или несколько символов (слов) при сравнении с эталоном. Все остальные символы, отличные от символов частичной обработки, в тексте обучаемого должны следовать в том же порядке, что и в эталоне ответа.

3. *Логический анализ.* Логический метод анализа дает возможность формирования ответа в свободно-конструируемой форме. В данном случае ответ может представлять собой фразу или предложение, в котором порядок слов строго не определен. В словах могут игнорироваться окончания или другие части.

Основным отличием данного метода анализа от анализа по ключевым словам является то, что исключается необходимость перечисления всех возможных последовательностей ключевых слов при рассмотрении многословных ответов, так как логический метод позволяет с помощью одного эталона проанализировать несколько вариантов ответов. Цель этой деятельности - преодоление чрезмерной заданности ответов обучаемого, что является общим недостатком многих ППС.

Недостатками такого рода программ являются

- снижение мотивации в ходе работы с программой;
- возникновение «провалов» (пробелов) в знаниях, связанных с произвольным рассеянием внимания в процессе работы с программой, а также ослаблением системного связывания знаний при отсутствии их интонационного выделения;
- сложность и высокая трудоемкость организации учебного диалога, а также диагностирующей и управляющей обучением части программы.

Ввиду чрезвычайно высокой трудоемкости написания программ такого рода на языках программирования и высоких требований к программистской квалификации разработчиков, они часто разрабатываются с использованием программных оболочек автоматизированных учебных курсов, имеющих свой язык программирования, интерфейс, рассчитанный на разработчика-непрограммиста.

Существует и продолжает разрабатываться большое количество инструментальных программ такого вида. Общим их недостатком является высокая трудоемкость разработки, затруднения организационного и методического характера при использовании в реальном учебном процессе школы. Организационные трудности связаны с тем, что такие программы невозможно исполь-

зовать в структуре урока из-за больших различий в темпе обучения разных учащихся. Методические трудности проявляются в том, что многие педагоги нередко склонны не соглашаться с методическими решениями и ходами при изложении теоретического материала, предложенными разработчиками инструментальной программы. В работе хорошего учителя много творческих, авторских моментов, в важности которых часто не отдают себе отчета создатели программ.

Программы 3-го типа (моделирующие) основаны на графически-иллюстративных возможностях компьютера, с одной стороны, и вычислительных, с другой, и позволяют осуществлять компьютерный эксперимент. Такие программы предоставляют ученику возможность наблюдать на экране дисплея некоторый процесс, влияя на его ход подачей команды с клавиатуры, меняющей значения параметров.

Программы 4-го типа (игры) предоставляют в распоряжение ученика некоторую воображаемую среду, существующий только в компьютере мир, набор каких-то возможностей и средств их реализации. Использование предоставляемых программой средств для реализации возможностей, связанных с изучением мира игры и деятельностью в этом мире, приводит к развитию обучаемого, формированию у него познавательных навыков, самостоятельному открытию им закономерностей, отношений объектов действительности, имеющих всеобщее значение.

Наибольшее распространение получили обучающие программы первых двух типов в связи с их относительно невысокой сложностью, возможностью унификации при разработке многих блоков программ. Если программы 3-го и 4-го типов требуют большой работы программистов, психологов, специалистов в области изучаемого предмета, педагогов-методистов, то технология создания программ 1-го и 2-го типов ныне сильно опростилась с появлением инструментальных средств или наполняемых автоматизированных обучающих систем (АОС).

Основные действия, выполняемые программами первых двух типов:

- предъявление кадра с текстом и графическим изображением;
- предъявление вопроса и меню вариантов ответа (или ожидание ввода открытого ответа);
- анализ и оценка ответа;

• предоставление кадра помощи при нажатии специальной клавиши. Они могут быть легко и унифицированно запрограммированы, так что разработчику обучающей программы остается ввести в компьютер только соответствующий текст, варианты ответов, нарисовать на экране с помощью манипулятора «мышь» картинку. Создание обучающей программы в этом случае выполняется совершенно без программирования, не требует серьезных компьютерных познаний и по силам любому педагогу-предметнику средней школы. Названия наиболее известных отечественных АОС: «Урок», «Адонис», «Магистр», «Stratum». Используются в России и зарубежные системы: «Linkway», «TeachCad» и др. Многие из этих систем имеют хорошие графические подсистемы и позволяют создавать не только статические картинки, но и динамические графические фрагменты в духе «мультимедиа» (речь об этом пойдет ниже).

Создание обучающей системы с использованием инструментальных программ обычно проходит четыре стадии.

1. Разработка сценария обучающей программы: на этой стадии педагог должен принять решение о том, какой раздел какого учебного курса он будет переводить в обучающую программу, продумать материал информационных кадров, такие вопросы и варианты ответов к ним, чтобы они диагностировали трудности, с которыми будут сталкиваться ученики при освоении материала, разработать схему прохождения программы, систему взаимосвязей между ее отдельными кадрами и фрагментами.

2. Ввод в компьютер текстов отдельных кадров будущей программы, рисование картинок, формирование контролирующих фрагментов: вопросов, вариантов ответов к ним и способов анализа правильности ответов. На этой стадии педагогу потребуется минимальное владение функциями компьютера и возможностями ввода и редактирования, встроенными в инструментальную программу.

3. Связывание отдельных элементов обучающей программы в целостную диалоговую систему, установление взаимосвязей между кадрами, вопросами и помощью, окончательная доводка программы.

4. Сопровождение программы во время ее эксплуатации, внесение в нее исправлений и дополнений, необходимость которых обнаруживается при ее использовании в реальном процессе обучения.

Очевидно, что создание обучающих программ средствами инструментальных систем может снять остроту главной проблемы компьютерного обучения -отсутствия в достаточном количестве и разнообразии качественных обучающих программ, так чтобы компьютерное обучение могло превратиться из жанра «показательных выступлений» на открытых уроках в действительно систематическое обучение учебным дисциплинам или их большим разделам.

В качестве первого шага к компьютерным технологиям обучения нужно рассматривать тренирующие и контролирующие программы. Нет ничего проще (с этой задачей могут справиться даже учащиеся старших классов, изучающие информатику), чем подготовить контролирующую программу по любому разделу любого учебного курса на языке программирования BASIC или с использованием инструментальных программ. Использовать такие контролирующие программы можно систематически. Это не потребует кардинальных изменений в существующем учебном процессе и избавит учителя от непроизводительных, рутинных операций по проверке письменных работ, контролю знаний учащихся, решит проблему накопляемости оценок. Из-за тотальности контроля учащиеся получают мощный стимул к обучению.

Следующая проблема компьютерного обучения связана с тем, что использование компьютера не вписывается в стандартную классно-урочную систему. Компьютер - это средство индивидуального обучения в условиях нелимитированного времени, и именно в этом качестве он должен использоваться. Соответствующие организационные формы учебного процесса и труда учителей еще предстоит найти и внедрить в практику. Важно, чтобы ученик при компьютерном обучении не был ограничен жесткими временными рамками, чтобы педагогу не надо было работать «на класс» в целом, а чтобы он мог пообщаться с каждым учеником, дать индивидуальную консультацию по работе с обучающей программой и по материалу, в ней содержащемуся, помочь преодолеть индивидуальные затруднения.

При проведении урока с использованием компьютеров работа педагога проходит фазы

- планирования урока (определяется место урока в системе занятий по данной дисциплине, время проведения в кабинете электронно-вычислительной техники, тип урока и его примерная структура, необходимые для его проведения программные средства);

- подготовки программных средств (наполнение оболочек контролирующих программ и обучающих систем соответствующими дидактическими материалами, подбор моделирующих программ, размещение программных средств на соответствующем магнитном диске, проверка запускаемости программ);

- проведения самого урока;

- подведения итогов (внесение исправлений в обучающие программы, архивирование их для будущего использования, обработка результатов компьютерного тестирования, удаление лишних временных файлов с магнитных дисков).

Отдельное направление использования компьютера в обучении - интегрирование предметных учебных курсов и информатики. При этом компьютер используется уже не как средство обучения, а как средство обработки информации, получаемой при изучении традиционных дисциплин - математики, физики, химии, экологии, биологии, географии. С помощью инструментальных программ на компьютере можно решать математические задачи в аналитическом виде, строить диаграммы и графики, проводить вычисления в табличном виде, готовить текст, схемы и т.д. Компьютер выступает при этом в качестве средства предметной деятельности, приближая стиль учебной деятельности на уроках к стандартам современной научной, технологической и управленческой деятельности.

Особые ожидания при таком использовании компьютера связываются с компьютерными телекоммуникациями, с возможностями локальных и глобальных компьютерных сетей. Весьма перспективной технологией обучения является метод групповых исследовательских проектов, моделирующий деятельность реального научного сообщества. Такая технология включает следующие моменты:

- первоначальную мотивацию исследования; обнаружение какого-либо парадокса, постановку проблемной задачи;

- поиск объяснения парадокса, построение гипотез;

- проведение исследований, экспериментов, наблюдений и измерений, литературных изысканий с целью доказать или отвергнуть гипотезы, объяснения;

- групповое обсуждение результатов, составление отчета, проведение научной конферен-

ции;

- решение вопроса о практическом применении результатов исследований; разработку и защиту итогового проекта по теме.

Работа над проектом продолжается от двух недель до двух месяцев. На заключительных стадиях работы над проектом обычно возникают новые проблемные задачи, обнаруживаются новые парадоксы, т.е. создается мотивация для осуществления новых проектов.

Использование компьютера очень хорошо вписывается в эту технологию обучения, особенно если имеется возможность реализовать компьютерные телекоммуникации: обмениваться сообщениями по электронной почте с классами в других городах и даже странах, параллельно выполняющими такой же проект. Телекоммуникационная составляющая проекта позволяет резко повысить интерес учащихся к выполнению проекта, делает естественным использование компьютера для представления результатов наблюдений и измерений, способствует формированию информационной культуры учащихся. Проекты, построенные на сопоставлении местных условий, изучении в них общего и особенного, прививают учащимся глобальное видение мира. Учебные телекоммуникационные проекты чрезвычайно популярны в Соединенных Штатах. Сотни таких проектов для десятков тысяч классов во всех странах мира проводят ежегодно многие глобальные компьютерные сети учебно-научного назначения. Имеется опыт использования телекоммуникационных проектов и в российских условиях.

Содержание обучения по методу проектов является межпредметным, интегрированным, привлекающим знания из различных областей, как и проблемы, возникающие на практике. Обучение по методу проектов, кроме изучения конкретных разделов наук, позволяет достигать и другие педагогические цели:

- развитие письменной речи;
- овладение компьютерной грамотностью, освоение текстового редактора, компьютерных телекоммуникационных программ;
- развитие общих навыков решения проблем;
- развитие навыков работы в группе;
- развитие навыков творческой работы.

В перспективе - развитие учебных курсов, использующих метод групповых проектов и компьютерные телекоммуникации, по разделам краеведения в географии и истории, по биологии и литературе, по иностранным языкам.

5.3. КОМПЬЮТЕРНОЕ ТЕСТИРОВАНИЕ

Обучение - многогранный процесс, и контроль знаний - лишь одна из его сторон. Однако именно в ней компьютерные технологии продвинулись максимально далеко, и среди них тестирование занимает ведущую роль. В ряде стран тестирование потеснило традиционные формы контроля - устные и письменные экзамены и собеседования.

По-видимому, многие преподаватели уже прошли через некоторую эйфорию при создании тестов и поняли, что это - весьма непростое дело. Куча бессистемно надерганных вопросов и ответов - далеко еще не тест. Оказывается, что для создания адекватного и эффективного теста надо затратить много труда. Компьютер может оказать в этом деле немалую помощь.

Существует специальная теория тестирования, оперирующая понятиями надежность, валидность, матрица покрытия и т.д., не специфических именно для компьютерных тестов. Здесь мы не будем в нее углубляться, сосредоточившись в основном на технологических аспектах.

Как отмечалось выше, широкое распространение в настоящее время получают инструментальные авторские системы по созданию педагогических средств: обучающих программ, электронных учебников, компьютерных тестов. Особую актуальность для преподавателей школ и вузов приобретают программы для создания компьютерных тестов - тестовые оболочки. Подобных программных средств существует множество, и программисты-разработчики готовы строить новые варианты, так называемых, авторских систем. Однако широкое распространение этих программных средств сдерживается отсутствием простых и нетрудоемких методик составления тестовых заданий, с помощью которых можно «начинять» оболочки. В настоящем разделе представлены некоторые подходы к разработке компьютерных тестов.

Технология проектирования компьютерных тестов предметной области. Экспертами

чаще используется метод нисходящего проектирования модели знаний (технология «сверху - вниз»). Вначале строится генеральное содержание предметной области с разбивкой на укрупненные модули (разделы). Затем проводится детализация модулей на элементарные подмодули, которые, в свою очередь, наполняются педагогическим содержанием.

Другой метод проектирования «снизу - вверх» (от частного к общему) в большинстве случаев реализуется группой экспертов для разработки модели знаний сложной и объемной предметной области или для нескольких, близких по структуре и содержанию, предметных областей.

Каждый модуль предполагает входящую информацию, состоящую из набора необходимых понятий из других модулей и предметных областей, а на выходе создает совокупность новых понятий, знаний, описанных в данном модуле, рис. 6.13.

Модуль может содержать подмодули. Элементарный подмодуль - неделимый элемент знания - может быть представлен в виде базы данных, базы знаний, информационной модели. Понятия и отношения между ними представляют семантический граф (рис. 6.14).

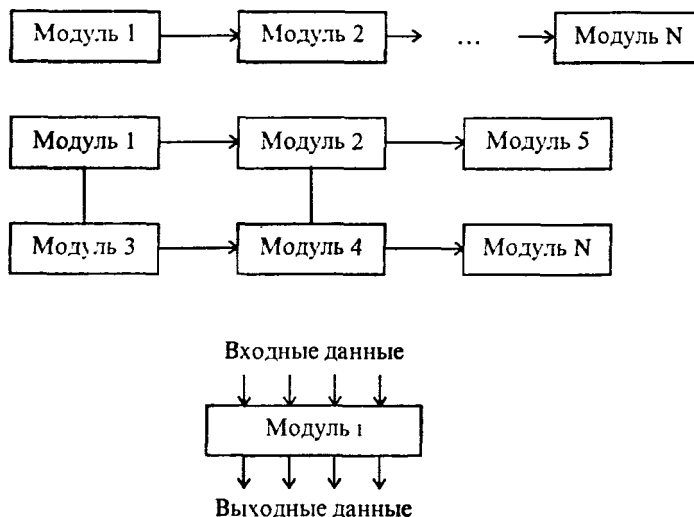


Рис. 6.13. Структуры линейной модели знаний

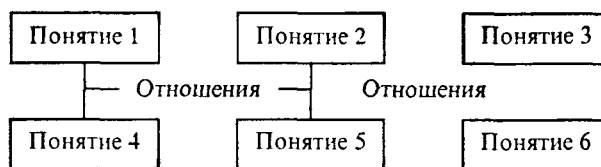


Рис. 6.14. Семантический граф модуля знаний

Приведем пример элемента модуля знаний по теме «Исследование графиков функций», рис. 6.15.



Рис. 6.15. Пример элемента модуля знаний

Модульное представление знаний помогает

- организовать четкую систему контроля с помощью компьютерного тестирования, поскольку допускает промежуточный контроль (тестирование) каждого модуля, итоговый контроль по всем модулям и их взаимосвязям, а также эффективно использовать методику «черного ящика»;

- осуществлять наполнение каждого модуля педагогическим содержанием;
- выявить и учитывать семантические связи модулей и их отношения с другими предметными областями.

Проектирование модели знаний играет важную роль. для образовательного процесса. От этого в конечном счете зависит обучающая среда: учитель с его квалификацией и опытом, средства и технологии обучения, а главное - контроль обучения.

Модульный принцип построения модели знаний позволяет использовать принцип исчерпывающего контроля - полный перебор всех тестовых заданий для заданной предметной области, что характерно для итоговых измерений уровня обученности.

Можно выделить два принципиальных способа контроля (тестирования) некоторой системы:

- 1) метод «белого ящика» - принцип тестирования экспертной модели знаний;
- 2) метод «черного ящика» - тестирование некоторой сложной системы по принципу контроля входных и выходных данных (наиболее подходит к компьютерному тестированию).

Для упрощения дальнейшего изложения введем ряд определений и понятий.

Тестирование - процесс оценки соответствия личностной модели знаний ученика экспертной модели знаний. Главная цель тестирования - обнаружение несоответствия этих моделей (а не измерение уровня знаний), оценка уровня их несоответствия. Тестирование проводится с помощью специальных тестов, состоящих из заданного набора тестовых заданий.

Тестовое задание - это четкое и ясное задание по предметной области, требующее однозначного ответа или выполнения определенного алгоритма действий.

Тест - набор взаимосвязанных тестовых заданий, позволяющих оценить соответствие знаний ученика экспертной модели знаний предметной области.

Тестовое пространство - множество тестовых заданий по всем модулям экспертной модели знаний.

Класс эквивалентности - множество тестовых заданий, таких, что выполнение учеником одного из них гарантирует выполнение других.

Полный тест - подмножество тестового пространства, обеспечивающее объективную оценку соответствия между личностной моделью и экспертной моделью знаний.

Эффективный тест - оптимальный по объему полный тест.

Самой сложной задачей эксперта по контролю является задача разработки тестов, которые позволяют максимально объективно оценить уровень соответствия или несоответствия личностной модели знаний ученика и экспертной модели.

Подбор тестовых заданий осуществляется экспертами-педагогами методологией «белого ящика», а их пригодность оценивают с помощью «черного ящика».

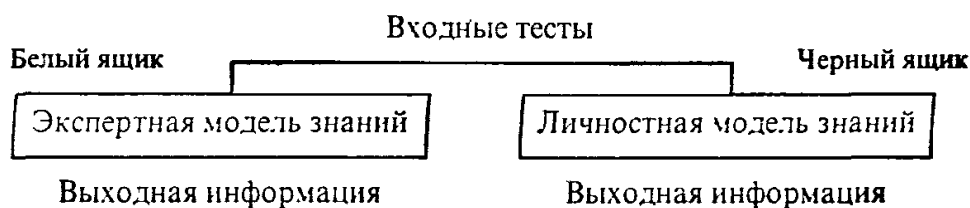


Рис. 6.16. Схема создания тестовых заданий

Самый простой способ составления тестовых заданий - формирование вопросов к понятиям, составляющим узлы семантического графа (рис. 6.16), разработка упражнений, требующих для их выполнения знания свойств выбранного понятия. Более сложным этапом является разработка тестовых заданий, определяющих отношения между понятиями. Еще более глубокий уровень заданий связан с их подбором, выявляющим связь понятий между отдельными модулями.

Множество тестовых заданий (тестовое пространство), вообще говоря, согласно принципу исчерпывающего тестирования, может быть бесконечным. Например, для исчерпывающего контроля знания таблицы умножения целых чисел от 1 до 100 необходимо использовать 100x100 всех возможных комбинаций двух чисел. А для всех натуральных чисел тестовое пространство становится бесконечным.

Однако в каждом реальном случае существует конечное подмножество тестовых заданий, использование которых позволяет с большой вероятностной точностью оценить соответствие зна-

ний ученика заданным критериям по экспертной модели знаний (полный тест).

Из полного теста можно выделить эффективный тест (оптимальный по объему набор тестовых заданий, гарантирующий оценку личностной модели ученика заданным критериям). Выбор эффективного теста зависит от удачного разбиения тестового пространства на классы эквивалентности, пограничные условия, создание тестов на покрытие путей и логических связей между понятиями и модулями.

В примере с таблицей умножения одним из классов эквивалентности может выступить множество заданий перемножения всех натуральных чисел на 1: $1*1$, $1*2$, $1*3$ и т.д. Поэтому в тест достаточно включить всего лишь несколько тестовых заданий из этого класса эквивалентности.

В дальнейшем необходим тестовый эксперимент на группе учащихся, который позволит провести корректировку и доводку теста до вида эксплуатации (методика «черного ящика»).

Таким образом, построение компьютерных тестов можно осуществлять по следующим последовательным шагам:

- 1) формализация экспертной целевой модели знаний;
- 2) нисходящее (или снизу - вверх) проектирование тестового пространства;
- 3) формирование и наполнение тестовых заданий;
- 4) формирование полного компьютерного теста;
- 5) тестовый эксперимент;
- 6) выбор эффективного теста;
- 7) анализ, корректировка и доводка теста до вида эксплуатации.

Типы компьютерных тестов. В соответствии с моделью знаний выделим три класса компьютерных тестов на знания, умения и навыки. Отметим, что типы компьютерных тестовых заданий определяются способами однозначного распознавания ответных действий тестируемого.

1. Типы тестовых заданий по блоку «знания»:

- вопросы альтернативные (требуют ответа да - нет);
- вопросы с выбором (ответ из набора вариантов);
- вопросы информативные на знание фактов (где, когда, сколько);
- вопросы на знание фактов, имеющих формализованную структуру (в виде информационной модели или схемы знаний);
- вопросы по темам, где имеются однозначные общепринятые знаковые модели; математические формулы, законы, предикатные представления, таблицы;
- вопросы, ответы на которые можно контролировать по набору ключевых слов;
- вопросы, ответы на которые можно распознавать каким-либо методом однозначно.

2. Типы тестовых заданий по блоку «навыки» (распознавание деятельности: манипуляции с клавиатурой; по конечному результату):

- задания на стандартные алгоритмы (альтернативные да — нет, выбор из набора вариантов);
- выполнение действия.

3. Типы тестовых заданий по блоку «умения». Те же самые, что навыки, но использующие нестандартные алгоритмы и задачи предметной области при контроле времени их решения:

- задания на нестандартные алгоритмы (альтернативные да - нет, выбор из набора вариантов);
- выполнение действия.

Выбор типов тестов определяется

- особенностями инструментальных тестовых программ (тестовыми оболочками);
- особенностями предметной области;
- опытом и мастерством экспертов.

Инструментальные тестовые оболочки. Для создания тестов по предметной области разработаны и разрабатываются специальные инструментальные программы-оболочки, позволяющие создавать компьютерные тесты путем формирования базы данных из набора тестовых заданий.

Инструментальные программы, позволяющие разрабатывать компьютерные тесты, можно разделить на два класса: универсальные и специализированные. Универсальные программы содержат тестовую оболочку как составную часть. Среди них «Адонис» (Москва), «Linkway» (Mi-

сrosoft), «Фея» (Томск), «Радуга» (Москва) и т.п. Специализированные тестовые оболочки предназначены лишь для формирования тестов. Это - «Аист» (Москва), «I_low» (Иркутск), «Тест» (Красноярск) и др.

Для того, чтобы разработать компьютерный вариант теста с помощью одной из названных выше программ, необходимо уяснить, какие формы тестовых заданий они допускают.

Как правило, компьютерные формы представления тестовых заданий могут выглядеть следующим образом.

1. Вопросы с фасетом. Задание вопроса, в котором меняются признаки.

Пример: Назовите столицу страны АНГЛИЯ : ? ____.

2. Вопросы с шаблоном ответа.

Пример: В каком году произошла Октябрьская революция? В ____ году.

3. Вопросы с набором ключевых слов (изображений, обозначений), из которых можно конструировать ответ.

Пример: Какие силы действуют на тело, движущееся по наклонной плоскости? (сила трения, сила упругости, сила тяжести, сила реакции опоры).

4. Закрытая форма вопроса: номер правильного ответа.

Пример: Какой климат в Красноярском крае?

1. Континентальный.

2. Субтропики.

3. Умеренный.

4. Резко-континентальный.

5. Задание на соответствие: несколько вопросов и несколько ответов.

Пример: а) Кто автор планетарной модели?

б) Кто автор закона тяготения?

в) Кто автор поэмы «Мцыри»?

а) М.Ю.Лермонтов

б) Э.Резерфорд

в) И. Ньютон

6. Конструирование ответа (шаблонный и бесшаблонный варианты): ответ формируется путем последовательного выбора элементов из инструментария по типу меню.

Пример: Чему равна производная функции $y = \sin(x) + \cos(x)$?

$y' = (\sin(x), \cos(x), \operatorname{tg}(x), +, -, /, *, \log(x), 1, 2, 3, 4, 5 \text{ и т.д.})$

7. Задание на конструирование изображений: с помощью графредактора, меню изображений (аналогично предыдущему примеру).

8. Задание на демонстрацию с движущимися объектами. Ответ - в виде действия тестируемого (определенный набор клавиш).

Пример: Клавиатурный тренажер на время.

Перечисленные формы компьютерного представления тестовых заданий не исчерпывают их многообразия. Многое зависит от мастерства и изобретательности эксперта по тестированию. При создании тестов важно учитывать многие обстоятельства: личность тестируемого, вид контроля, методику использования тестов в учебном процессе и т.п.

Хорошим считается тест, если

- он восприимчив к угадыванию тестируемым;
- он восприимчив к невнимательности и ошибочным действиям тестируемого;
- он положительно влияет на тестируемого и педагога, который использует тест. При этом тест используется обучаемым как
- обучение (тренажер, самоконтроль);
- контроль.

Для учителя тест служит

- корректировке учебного процесса;
- использованию как вспомогательного средства для контроля (текущего);
- использованию как дидактического средства для обучения;
- для дистанционного обучения.

Пример теста по школьному курсу информатики. В 1996 г. Республиканский центр тестирования использовал тесты по некоторым школьным предметам, в частности по информатике.

Ниже приводится один из его вариантов (разработчики: Н. Г. Граве, И.А.Елисеев, Г.В.Тюрникова).
Тесты построены на основе канонического принципа: вопрос и варианты ответа.

Разработчиками выбрана следующая модель знаний школьного курса информатики:

Модуль 1. ВВЕДЕНИЕ

1. Измерение информации
2. Свойство информации
3. Измерение информации
4. Предмет информации. Фундаментальные понятия
5. История развития вычислительной техники

Модуль 2. УСТРОЙСТВО И РАБОТА ЭВМ

6. Состав информационно-измерительного комплекса
7. Поколения ЭВМ
8. Арифметические основы ЭВМ
9. Состав информационно-измерительного комплекса
10. Арифметические основы ЭВМ 1
11. Физические основы ЭВМ
12. Состав информационно-измерительного комплекса

Модуль 3. АЛГОРИТМИЗАЦИЯ

13. Величины, тип, имя, значения, вид
14. Величины, тип, имя, значения, вид
15. Величины, тип, имя, значения, вид
16. Типы алгоритма
17. Способы описания
18. Способы описания
19. Алгоритм, свойства
- 20 - 24. Остальные вопросы как единый подраздел

Модуль 4. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

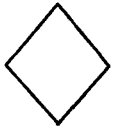
- 25 - 28. Операционные системы
- 29 - 30. Текстовый, графический, музыкальный редакторы
- 31 - 32. Базы данных
33. Электронные таблицы

Модуль 5. ЗАКЛЮЧЕНИЕ

- 34 - 36. Перспективы развития

Российский тест по информатике N 01

01. кбайт-это
 - 1) 1000 символов, 2) 1024 байт, 3) 8 бит, 4) 1000 байт
02. Достоверность - это свойство
 - 1) алгоритма, 2) компьютера, 3) информации, 4) языка программирования
03. Наибольший объем памяти требуется для хранения
 - 1) «10», 2) 10, 3) «десять», 4) (10)
04. Носителем информации является
 - 1) провода, 2) принтер, 3) классный журнал, 4) телефон
05. Первая машина, автоматически выполняющая все 10 команд, была
 - 1) машина С.А.Лебедева, 2) машина Ч.Бэббиджа, 3) абак, 4) Pentium
06. Минимально необходимый набор устройств для работы компьютера содержит
 - 1) принтер, системный блок, клавиатуру;
 - 2) процессор, ОЗУ, монитор, клавиатуру;
 - 3) монитор, винчестер, клавиатуру, процессор;
 - 4) системный блок, дисководы, мышь

07. Элементной базой ЭВМ третьего поколения являются
1) ЭЛТ (электронно-лучевая трубка), 2) светодиоды.
3) ИС (интегральные схемы), 4) транзисторы
08. Число 3210-это
1)1000002, 2)358, 3)2116, 4)100001
09. К внешним запоминающим устройствам относится
1) процессор, 2) дискета, 3) монитор, 4) жесткий диск
10. Определить сумму трех чисел: $0012 + 0178 + 1112$
1)02310, 2)00910. 3)1112, 4)10002
11. Перевести число 3210 в двоичную систему счисления
1)100000, 2)111111, 3)101010, 4)100001
12. К внутренним запоминающим устройствам относится
1) монитор, 2) жесткий диск, 3) оперативная память (RAM), 4) флоппи-диск
13. Неверно записанное выражение
1)+3, 2) $\text{tg}(+3)$, 3)- $\text{tg}(-3)+1$, 4)- $\sin(-3)+(1)*(\text{tg}(+1))$
14. По выполнении следующего алгоритма $x := 7$; $y := 12+5$; $y := y + y - x$ значение x будет
1)7, 2)89, 3)94, 4)47
15. Если исполнить $X:=2$; $Y:=X+3$; $X:=X+1$; $Y:=X+3*Y$, то значение Y равно
1)0, 2)-10, 3)18, 4)6.5
16. При $t \gg 17$ будет ложно
1) $t=17,01$, 2) $t \gg 212$ и $t \gg 1000$, 3) $t=17$, 4) $t \gg 17$ и $t \ll 20$
17. Каким должно быть значение k во фрагменте алгоритма
 $a:=1$ нц
 для i от 2 до k $a:=a*i$; $i:=i+1$
 кц
 чтобы a стало равно 11?
 1) 8, 2) 9, 3) 10, 4) 11
18. Для вывода данных в блок-схемах используют фигуру
 1)  2)  3)  4) 
19. Геометрическая фигура  используется в блок-схемах для обозначения
1) условия, 2) останова, 3) любого действия, 4) цикла «для»
20. Не является свойством алгоритма
1) универсальность, 2) массовость, 3) результативность, 4) дискретность
21. При составлении алгоритма для вычисления функции $y=a*\sin(x)$ аргументами являются
1) $\sin$; 2) a, x ; 3) x ; 4) x, y
22. Сколько раз выполнится цикл:
 $i:=1$; $a:=10$; $n:=2$;
 нц пока $a>0$ $a:=a-n*i$
 кц
 1) 0, 2) 10, 3) 5, 4) 4
23. В качестве имени переменной может быть
1) 1996, 2) a1996, 3) 1996a, 4) -1996
24. Для описания циклического алгоритма используется конструкция
1)ПОКА, 2) ЕСЛИ, 3) ВЫБОР, 4) ПРОЦЕДУРА
25. Какая программа является интерпретатором команд MS-DOS ?
1) AUTOEXEC. BAT, 2) MSDOS.SYS,
3) CONFIG.SYS, 4) CQMMAND.COM
26. Минимально необходимый набор файлов для работы компьютера в MS-DOS
1) IO.SYS, MSDOS.SYS
2) IO.SYS, MSDOS.SYS, COMMAND.COM
3) IO.SYS, MSDOS.SYS, COMMAND.COM, CONFIG.SYS
4) IO.SYS, MSDOS.SYS, COMMAND.COM .AUTOEXEC.BAT

27. Сколько символов в своем полном имени может содержать директория
1)11, 2)8, 3)7, 4)12
28. Неверным будет утверждение
1) файл с расширением .TXT может быть не текстовым
2) системный диск может не содержать файл CONFIG.SYS
3) файл AUTOEXEC.BAT может не содержать ни одной строки
(ни одного байта)
4) файл должен содержать в расширении не менее трех букв
29. Текстовый редактор «Лексикон» - это
1) прикладная программа 2) базовое программное обеспечение
3) сервисная программа 4) редактор шрифтов
30. Под термином «интерфейс» понимается
1) внешний вид программной среды, служащий для обеспечения диалога с пользователем
2) связь текстового редактора с устройством печати
3) совокупность файлов, содержащихся в одном каталоге
4) устройство хранения графической информации
31. База данных - это
1) текстовый редактор 2) совокупность связанных между собой сведений
3) операционная оболочка 4) утилиты NC
32. Графический редактор нужен для
1) нормальной работы баз данных 2) быстрого поиска информации
3) проигрывания звуковых файлов 4) создания рисунков
33. В отличие от бумажных табличных документов, электронные таблицы обычно
1) имеют большую размерность
2) позволяют быстрее производить расчеты
3) обладают всеми свойствами, перечисленными в пунктах 1 - 2
4) стоят дороже
34. Что делает невозможным подключение компьютера к глобальной сети ?
1) тип компьютера
2) состав периферийных устройств
3) отсутствие винчестера
4) отсутствие телефона
35. Дан E-mail: artem@svremech.msk.ru. Слово msk означает
1) город назначения 2) тип компьютера 3) каталог 4) имя пользователя
36. Первый PHOTO CD был произведен фирмой
1)IBM, 2) APPLE, 3) KODAK, 4) POLAROID

Ответы на тестовые задания

1.-2); 2.-3); 3.-3); 4.-3); 5.-2); 6.-2); 7.-3); 8.-4); 9.-2);
10.-4); 11.-4); 12.-2); 13.-4); 14.-1); 15.-3); 16.-3); 17.-4); 18.-3);
19.-2); 20.-1); 21.-3); 22.-3); 23.-2); 24.-1); 25.-4); 26.-2); 27.-2);
28.-4); 29.-1); 30.-1); 31.-2); 32.-4); 33.-3); 34.-4); 35.-1); 36.-1).

5.4. ПЕРСПЕКТИВНЫЕ ИССЛЕДОВАНИЯ В ОБЛАСТИ КОМПЬЮТЕРНОГО ОБУЧЕНИЯ

Современные исследования в области применения компьютеров в обучении развиваются, в основном, в рамках нескольких основных направлений, которые можно обозначить следующим образом:

- 1) интеллектуальные обучающие системы;
- 2) учебные мультимедиа и гипермедиа;
- 3) учебные среды, микромиры и моделирование;
- 4) использование компьютерных сетей в образовании;

5) новые технологии для обучения конкретным дисциплинам.

Остановимся на некоторых из этих направлений подробнее

Интеллектуальные обучающие системы. Наиболее перспективным направлением развития систем компьютерного обучения является технология искусственного интеллекта (ИИ). Системы, использующие методику ИИ, называют интеллектуальными обучающими системами (ИОС). ИОС реализует адаптивное и двухстороннее взаимодействие, направленное на эффективную передачу знаний. Под адаптивностью понимается то, что система дает пояснения, подходящие каждому обучаемому, с помощью динамического управления, зависящего от процесса обучения. Двухстороннее взаимодействие - это взаимодействие со смешанной инициативой, при которой обучаемый может задать вопросы или просить систему решить задачу. ИОС отличаются друг от друга прежде всего методологиями представлений знаний о предметной области, об обучаемом и о процессе обучения.

Наиболее перспективным путем развития ИОС является, по-видимому, путь создания самообучающихся систем, приобретающих знания в диалоге с человеком. Общая архитектура системы совместного обучения человека и компьютера может определяться следующими компонентами;

- микромир;
- учащийся-человек;
- учащийся-компьютер;
- интерфейс между двумя учащимися и микромиром;
- интерфейс между двумя учащимися.

В основе разработки компьютерного «соученика» в центре внимания должно быть соотношение между управлением и коммуникацией. Прототипом такого рода системы можно считать MEMOLAB - обучающую среду с искусственным интеллектом по методологии экспериментальной психологии и человеческой памяти.

Другое направление развития систем искусственного интеллекта - распределенные системы, связывающие два и более компьютеров так, что ученики могут обучаться, сотрудничая или соревнуясь, каждый на своем компьютере. В этом случае возникает некое подобие «классного» обучения, но на совершенно ином уровне. Эксперименты и оценки показывают, что такое обучение оказывается более эффективным и интересным, чем обучение в одиночку.

Недостатком многих существующих ИОС является ориентация на специальные знания в рамках определенного предмета, так что в них не предусмотрена возможность простой адаптации к другой предметной области. Более общий подход состоит в развитии интеллектуального окружения (оболочки), из которого затем можно получить много ИОС путем наполнения различным содержанием, как баз знаний. Пример такой системы - EEPS, обучающая среда для решения задач, обеспечивающая обучение решению задач в качественных областях науки.

Система реализует модель преподавания, основанную на трех режимах:

- режим вопросов (обучаемый расспрашивает компьютер, с целью получения ответов на задачи и их объяснений);
- режим исследования (решения задачи совместными усилиями обучаемого с компьютером, обучаемый предоставляет требуемую информацию для решения задачи);
- режим решения (обучаемый решает задачу самостоятельно, получая минимальную помощь и советы компьютера).

Система диагностики представляет стратегию решения задач студентом в виде одного из следующих стилей:

- дефектный стиль (студент, зная материал, допускает одну или более концептуальных ошибок);
- стиль «вокруг да около» (студент пытается найти решение многими неверными путями, задает много не относящихся к делу вопросов);
- рефлексивный стиль (когда студент знает материал, но решает задачу постепенно, иногда проходя через множество промежуточных этапов);
- импульсивный стиль (когда студент спешит прийти к заключению без достаточных оснований);
- смешанный стиль - комбинация двух или более перечисленных выше стилей.

Основанные на знаниях модели обучаемых могут быть построены с использованием раз-

личных видов дифференциального анализа, когнитивной диагностики.

В современных интеллектуальных обучающих системах, в основном, используются знания о качественных (количественных) аспектах процесса обучения. Однако, необходимо учитывать и мотивационную сторону обучения. Мотивационные аспекты обучения можно классифицировать в соответствии с такими явлениями, как соревновательность, заинтересованность, самоконтроль, уверенность и удовлетворение.

Обучающая система должна

- определять мотивационное состояние обучаемого;
- реагировать с целью мотивации рассеянных, менее уверенных или недовольных учеников или поддержки уже мотивированных учеников. Примеры мотивационной тактики:
 - если менее вверенный ученик правильно решает задачу, система может предложить ему подобную задачу для закрепления;
 - внимание рассеянных или неактивных обучаемых может быть привлечено неожиданными эффектами или вводными комментариями;
 - интерес может быть повышен головоломками, вопросами или знакомством с новыми темами.

Учебная мультимедиа и гипермедиа представляет собой развитие технологии программированного обучения, хотя упор делается не на адаптивность обучения и его методическое обоснование, а на внешнюю иллюстративно-наглядную сторону. Современные графические и звуковые возможности компьютера, а также возможность комплексирования его в качестве управляющего устройства с системами учебного телевидения, обусловили появление средств гипер- и мультимедиа. Научные исследования в данной области связаны с разработкой технологий создания учебных курсов большего размера на основе возможностей мультимедиа. Под управлением компьютера система мультисред может производить в едином представлении объединение текста, графики, звуков, видео-образов и мультипликации. Технология мультимедиа в последнее время широко применяется для создания электронных книг (и учебников).

Развитием идей мультимедиа являются технологии компьютерной виртуальной реальности. В этом случае с помощью специальных экранов, датчиков, шлемов, перчаток и т.п. полностью моделируется управление, например, самолетом, так что у обучаемого возникает полная иллюзия того, что он находится в кабине самолета и им управляет.

Таковы основные направления исследований в области компьютерного обучения и основные подходы в компьютерном обучении. Ситуация, сложившаяся в области компьютерного обучения, является парадоксальной: несмотря на активно и в различных направлениях ведущиеся поиски, обилие результатов, зреет ощущение необходимости кардинальных изменений концепции обучения, глубинного изменения подхода к компьютерному обучению. В первую очередь, требуется разработка адекватной теории компьютерного обучения, новых методов представлений знаний и моделирования процесса обучения и поведения обучаемого.

Компьютерное обучение остается очень интересной и перспективной областью исследований, привлекающей передовых ученых, педагогов и методистов всего мира. С внедрением компьютерного обучения стали меняться стили и устоявшиеся подходы к обучению, стала быстро меняться сама эта традиционная сфера человеческой деятельности. Трудно переоценить значение и влияние этих изменений на судьбы человеческой цивилизации в целом.

Контрольные вопросы и задания

1. Разработайте модель знания по школьному разделу «действия с дробями», используя модульный принцип.
2. Разработайте тест на знание таблицы умножения чисел от 0 до 100.
3. Используя какую-либо инструментальную тестовую оболочку, разработайте компьютерный тест по тестовым заданиям курса информатики, описанным в настоящей главе.

Дополнительная литература к главе 6

1. Балыко Г. Г., Пугач В. И., Фишман Л. И. Управление школой и базы данных. -Самара: СГПИ, 1992.

2. *Бережанский Д. Г.* Практическое программирование на dBASE. - М.: Финансы и статистика. 1989.
3. *Герман О. В.* Введение в теорию экспертных систем и обработку знаний: - Минск: «Дизайн-ПРО», 1995.
3. *Глушков В. М.* Основы безбумажной информатики / Изд. 2-е. - М.: Наука, 1987.
4. *А.Каратыгин С., Тихонов А., Долголаптев В.* Базы данных: простейшие средства обработки информации, электронные таблицы, системы управления базами данных. В 2-х томах. - М.: АБФ. 1995.
5. *Коновалова Н. В., Капралов Е. Г.* Введение в ГИС. - Петрозаводск: Петрозаводский гос-университет, 1995.
6. *Крамм Р.* Системы управления базами данных dBASE II и dBASE III для персональных компьютеров. - М.: Финансы и статистика, 1988.
7. САПР. Системы автоматизированного проектирования / Под ред. И. П. Норенкова. - Минск: Высшая школа, 1987.
8. *Свириденко С. С.* Современные информационные технологии. - М.: Радио и связь, 1989.
9. *Советов Б. Я.* АСУ. Введение в специальность. - М.: Высшая школа, 1989.
10. *Советов Б. Я.* Информационная технология. - М.: Высшая школа, 1992.
11. *Фурунжиев Р. М., Гугля В. А.* САПР, или как ЭВМ помогает конструктору. - Минск: Высшая школа, 1987.
12. Электронная почта в системе MS-DOS. Официальное руководство компании Редком.- М., 1995.
13. Журнал «Информатика и образование», с 1992 г.
14. Журнал «Педагогическая информатика», с 1994 г.

ГЛАВА 7

КОМПЬЮТЕРНОЕ МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ

ВВЕДЕНИЕ

Настоящая глава посвящена той из компьютерных технологий обработки информации, ради которой когда-то создали первую ЭВМ и ради которой сегодня в значительной мере создают супер-ЭВМ - решению прикладных научно-технических задач, среди которых задачи математического моделирования составляют видную долю.

Абстрактное моделирование с помощью компьютеров - вербальное, информационное, математическое - в наши дни стало одной из информационных технологий, в познавательном плане исключительно мощной. Изучение компьютерного математического моделирования открывает широкие возможности для осознания связи информатики с математикой и другими науками - естественными и социальными.

В данной главе, в значительной степени на примерах моделей из разных областей знания, показаны некоторые типичные задачи компьютерного математического моделирования. Их решение способствует выработке тех навыков, которые необходимы специалисту в области информатики.

Отметим, что, говоря о математических моделях, мы имеем в виду сугубо прикладной аспект. В современной математике есть достаточно формализованный подход к понятию «математическая модель». Внутри него вполне допустимо игнорировать вопрос о связи математики с реалиями физического мира. В этом подходе моделями являются, например, система целых чисел, система действительных чисел, евклидова геометрия, алгебраическая группа, топологическое пространство и т.д. К исследованию таких формальных моделей вполне можно подключить компьютеры, но все равно это останется «чистой» математикой. В данной главе термин «математическая модель» увязывается с некоторой предметной областью, сущностью окружающего мира.

Компьютерное математическое моделирование в разных своих проявлениях использует практически весь аппарат современной математики.

В данной главе предполагается знание основ математики:

- теории дифференциальных уравнений;

- аппроксимации функций (включая интерполяцию и среднеквадратичные приближения);
 - аналитической геометрии на плоскости и в пространстве;
 - математической статистики;
 - численных методов:
- а) решения алгебраических и трансцендентных уравнений;
 - б) решения систем линейных алгебраических уравнений;
 - в) интегрирования обыкновенных дифференциальных уравнений и их систем (задача Коши).

В тех немногих случаях, когда используемый математический аппарат выходит за пределы объема, традиционно считающегося достаточным для подготовки специалиста по информатике, минимально необходимые сведения приводятся в тексте.

§ 1. О РАЗНОВИДНОСТЯХ МОДЕЛИРОВАНИЯ

С понятием «модель» мы сталкиваемся с детства. Игрушечный автомобиль, самолет или кораблик для многих были любимыми игрушками, равно как и плюшевый медвежонок или кукла. В развитии ребенка, в процессе познания им окружающего мира, такие игрушки, являющиеся, по существу, моделями реальных объектов, играют важную роль. В подростковом возрасте для многих увлечение авиамоделированием, судомоделированием, собственноручным созданием игрушек, похожих на реальные объекты, оказало влияние на выбор жизненного пути.

Что же такое модель? Что общего между игрушечным корабликом и рисунком на экране компьютера, изображающим сложную математическую абстракцию? И все же общее есть: и в том, и в другом случае мы имеем образ реального объекта или явления, «заместителя» некоторого «оригинала», воспроизводящего его с той или иной достоверностью и подробностью. Или то же самое другими словами: модель является представлением объекта в некоторой форме, отличной от формы его реального существования.

Практически во всех науках о природе, живой и неживой, об обществе, построение и использование моделей является мощным орудием познания. Реальные объекты и процессы бывают столь многогранны и сложны, что лучшим способом их изучения часто является построение модели, отображающей лишь какую-то грань реальности и потому многократно более простой, чем эта реальность, и исследование вначале этой модели. Многовековой опыт развития науки доказал на практике плодотворность такого подхода.

В моделировании есть два заметно разных пути. Модель может быть похожей копией объекта, выполненной из другого материала, в другом масштабе, с отсутствием ряда деталей. Например, это игрушечный кораблик, самолетик, домик из кубиков и множество других натуральных моделей. Модель может, однако, отображать реальность более абстрактно - словесным описанием в свободной форме, описанием, формализованным по каким-то правилам, математическими соотношениями и т.д.

В прикладных областях различают следующие виды абстрактных моделей;

I) традиционное (прежде всего для теоретической физики, а также механики, химии, биологии, ряда других наук) математическое моделирование без какой-либо привязки к техническим средствам информатики;

II) информационные модели и моделирование, имеющие приложения в информационных системах;

III) вербальные (т.е. словесные, текстовые) языковые модели;

IV) информационные (компьютерные) технологии, которые надо делить

А) на инструментальное использование базовых универсальных программных средств (текстовых редакторов, СУБД, табличных процессоров, телекоммуникационных пакетов);

Б) на компьютерное моделирование, представляющее собой

- вычислительное (имитационное) моделирование;
- «визуализацию явлений и процессов» (графическое моделирование);
- «высокие» технологии, понимаемые как специализированные прикладные технологии, использующие компьютер (как правило, в режиме реального времени) в сочетании с измерительной аппаратурой, датчиками, сенсорами и т.д.

Итак, укрупненная классификация абстрактных (идеальных) моделей такова.

1. Вербальные (текстовые) модели. Эти модели используют последовательности предложений на формализованных диалектах естественного языка для описания той или иной области действительности (примерами такого рода моделей являются милицкий протокол, правила дорожного движения).

2. Математические модели - очень широкий класс знаковых моделей (основанных на формальных языках над конечными алфавитами), широко использующих те или иные математические методы. Например, можно рассмотреть математическую модель звезды. Эта модель будет представлять собой сложную систему уравнений, описывающих физические процессы, происходящие в недрах звезды. Математической моделью другого рода являются, например, математические соотношения, позволяющие рассчитать оптимальный (наилучший с экономической точки зрения) план работы какого-либо предприятия.

3. Информационные модели - класс знаковых моделей, описывающих информационные процессы (возникновение, передачу, преобразование и использование информации) в системах самой разнообразной природы.

Граница между вербальными, математическими и информационными моделями может быть проведена весьма условно; вполне возможно считать информационные модели подклассом математических моделей. Однако, в рамках информатики как самостоятельной науки, отделенной от математики, физики, лингвистики и других наук, выделение информационных моделей в отдельный класс является целесообразным.

Отметим, что существуют и иные подходы к классификации абстрактных моделей; общепринятая точка зрения здесь еще не установилась. В частности, есть тенденция резкого расширения содержания понятия «информационная модель», при котором информационное моделирование включает в себя и вербальные, и математические модели.

Основное содержание данной главы связано с прикладными математическими моделями, в реализации которых используются компьютеры. Это вызвано тем, что внутри информатики именно компьютерное математическое и компьютерное информационное моделирование могут рассматриваться как ее составные части. Компьютерное математическое моделирование связано с информатикой технологически; использование компьютеров и соответствующих технологий обработки информации стало неотъемлемой и необходимой стороной работы физика, инженера, экономиста, эколога, проектировщика ЭВМ и т.д. Неформализованные вербальные модели не имеют столь явно выраженной привязки к информатике - ни в принципиальном, ни в технологическом аспектах.

§2. ПОНЯТИЕ О КОМПЬЮТЕРНОМ МАТЕМАТИЧЕСКОМ МОДЕЛИРОВАНИИ

2.1. МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ И КОМПЬЮТЕРЫ

Математическая модель выражает существенные черты-объекта или процесса языком уравнений и других математических средств. Собственно говоря, сама математика обязана своим существованием тому, что она пытается отразить, т.е. промоделировать, на своем специфическом языке закономерности окружающего мира.

Путь математического моделирования в наше время гораздо более всеобъемлющ, нежели моделирования натурального. Огромный толчок развитию математического моделирования дало появление ЭВМ, хотя сам метод зародился одновременно с математикой тысячи лет назад.

Математическое моделирование как таковое отнюдь не всегда требует компьютерной поддержки. Каждый специалист, профессионально занимающийся математическим моделированием, делает все возможное для аналитического исследования модели. Аналитические решения (т.е. представленные формулами, выражающими результаты исследования через исходные данные) обычно удобнее и информативнее численных. Возможности аналитических методов решения сложных математических задач, однако, очень ограничены и, как правило, эти методы гораздо сложнее численных. В данной главе доминируют численные методы, реализуемые на компьютерах. Это связано с тем, что моделирование здесь рассматривается под углом зрения компьютерных (информационных) технологий. Такой подход несколько сужает возможности метода в целом; его

достоинство - некоторое снижение барьера необходимой математической подготовки (хотя, разумеется, и в численные методы при профессиональном занятии математическим моделированием приходится углубляться настолько, что при этом требуется значительное математическое образование). Наконец, отметим, что понятия «аналитическое решение» и «компьютерное решение» отнюдь не противостоят друг другу, так как

а) все чаще компьютеры при математическом моделировании используются не только для численных расчетов, но и для аналитических преобразований;

б) результат аналитического исследования математической модели часто выражен столь сложной формулой, что при взгляде на нее не складывается восприятия описываемого ей процесса. Эту формулу (хорошо еще, если просто формулу!) нужно протабулировать, представить графически, проиллюстрировать в динамике, иногда даже озвучить, т.е. проделать то, что называется «визуализацией абстракций». При этом компьютер - незаменимое техническое средство.

2.2. ЭТАПЫ И ЦЕЛИ КОМПЬЮТЕРНОГО МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ

Здесь мы рассмотрим процесс компьютерного математического моделирования, включающий численный эксперимент с моделью (рис. 7.1).

Первый этап - определение целей моделирования. Основные из них таковы:

1) модель нужна для того, чтобы понять как устроен конкретный объект, какова его структура, основные свойства, законы развития и взаимодействия с окружающим миром (**понимание**);

2) модель нужна для того, чтобы научиться управлять объектом (или процессом) и определить наилучшие способы управления при заданных целях и критериях (**управление**);

3) модель нужна для того, чтобы прогнозировать прямые и косвенные последствия реализации заданных способов и форм воздействия на объект (**прогнозирование**).

Поясним это на примерах. Пусть объект исследования - взаимодействие потока жидкости или газа с телом, являющимся для этого потока препятствием. Опыт показывает, что сила сопротивления потоку со стороны тела растет с ростом скорости потока, но при некоторой достаточно высокой скорости эта сила скачком уменьшается с тем, чтобы с дальнейшим увеличением скорости снова возрасти. Что же произошло, обусловив уменьшение силы сопротивления? Математическое моделирование позволяет получить четкий ответ: в момент скачкообразного уменьшения сопротивления вихри, образующиеся в потоке жидкости или газа позади обтекаемого тела, начинают отрываться от него и уноситься потоком.

Пример совсем из другой области: мирно сосуществовавшие со стабильными численностями популяции двух видов особей, имеющих общую кормовую базу, «вдруг» начинают резко менять численность - и здесь математическое моделирование позволяет (с известной долей достоверности) установить причину (или, по крайней мере, опровергнуть определенную гипотезу).

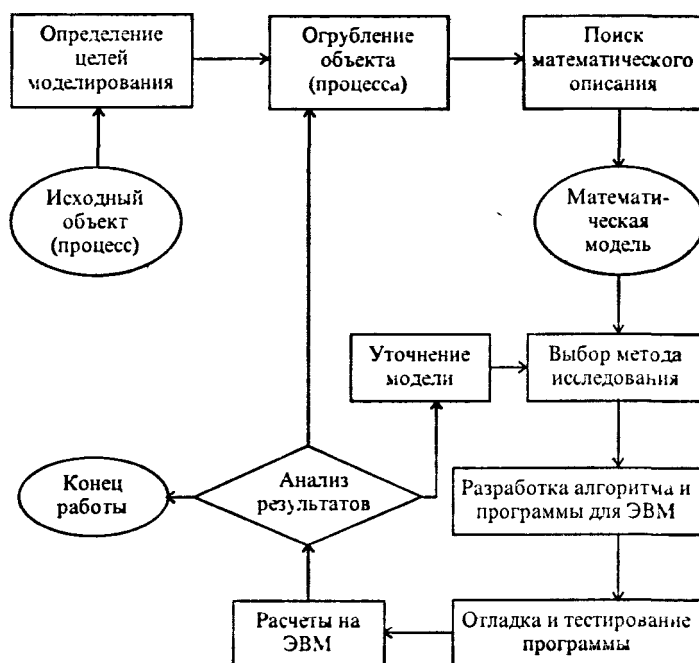


Рис. 7.1. Общая схема процесса компьютерного математического моделирования

Выработка концепции управления объектом - другая возможная цель моделирования. Какой режим полета самолета выбрать для того, чтобы полет был вполне безопасным и экономически наиболее выгодным? Как составить график выполнения сотен видов работ на строительстве большого объекта, чтобы оно закончилось в максимально короткий срок? Множество таких проблем систематически возникает перед экономистами, конструкторами, учеными.

Наконец, прогнозирование последствий тех или иных воздействий на объект может быть как относительно простым делом в несложных физических системах, так и чрезвычайно сложным - на грани выполнимости - в системах биолого-экономических, социальных. Если относительно легко ответить на вопрос об изменении режима распространения тепла в тонком стержне при изменениях в составляющем его сплаве, то несравненно труднее проследить (предсказать) экологические и климатические последствия строительства крупной ГЭС или социальные последствия изменений налогового законодательства. Возможно, и здесь методы математического моделирования будут оказывать в будущем более значительную помощь.

Составим список величин, от которых зависит поведение объекта или ход процесса, а также тех величин, которые желательно получить в результате моделирования. Обозначим первые (входные) величины через $x_1, x_2, \dots, x_n$; вторые (выходные) через $y_1, y_2, \dots, y_k$. Символически поведение объекта или процесса можно представить в виде

$$y_j = F_j(x_1, x_2, \dots, x_n) \quad (j=1, 2, \dots, k), \quad (7.1)$$

где F_j - те действия, которые следует произвести над входными параметрами, чтобы получить результаты. Хотя запись $F(x_1, x_2, \dots, x_n)$ напоминает о функции, мы здесь используем ее в более широком смысле. Лишь в простейших ситуациях $F(x)$ есть функция в том смысле, который вкладывается в это понятие в учебниках математики; чтобы это подчеркнуть, лучше использовать по отношению к $F(x)$ термин «оператор».

Входные параметры x_i могут быть известны «точно», т.е. поддаваться (по крайней мере, в принципе) измерению однозначно и с любой степенью точности - тогда они являются детерминированными величинами. Так, в классической механике, сколь сложной ни была бы моделируемая система, входные параметры детерминированы - соответственно, детерминирован, однозначно развивается во времени процесс эволюции такой системы. Однако, в природе и обществе гораздо чаще встречаются процессы иного рода, когда значения входных параметров известны лишь с определенной степенью вероятности, т.е. эти параметры являются вероятностными (стохастическими), и, соответственно, таким же является процесс эволюции системы (случайный процесс).

«Случайный» - не значит «непредсказуемый»; просто характер исследования, задаваемых вопросов резко меняется (они приобретают вид «С какой вероятностью...», «С каким математиче-

ским ожиданием...» и т.п.). Примеров случайных процессов не счесть как в науке, так и в обыденной жизни (силы, действующие на летящий самолет в ветренную погоду, переход улицы при большом потоке транспорта и т.д.).

Для стохастической модели выходные параметры могут быть как величинами вероятностными, так и однозначно определяемыми. Пример последнего: на перекрестке улиц можно ожидать зеленого сигнала светофора и полминуты, и две минуты (с разной вероятностью), но среднее время ожидания есть величина вполне определенная, и именно она может быть объектом моделирования.

Важнейшим этапом моделирования является разделение входных параметров по степени важности влияния их изменений на выходные. Такой процесс называется ранжированием (разделением по рангам). Чаще всего невозможно (да и не нужно) учитывать все факторы, которые могут повлиять на значения интересующих нас величин y_j . От того, насколько умело выделены важнейшие факторы, зависит успех моделирования, быстрота и эффективное гь достижения цели. Выделить более важные (или, как говорят, значимые) факторы и отсеять менее важные может лишь специалист в той предметной области, к которой относится модель. Так, опытный учитель знает, что на успех контрольной работы влияет степень знания предмета и психологический настрой класса; однако, влияют и другие факторы - например, каким уроком по счету идет контрольная, какова в этот момент погода и т.д. -фактически проведено ранжирование.

Отбрасывание (по крайней мере при первом подходе) менее значимых факторов огрубляет объект моделирования и способствует пониманию его главных свойств и закономерностей. Умело ранжированная модель должна быть адекватна исходному объекту или процессу в отношении целей моделирования. Обычно определить адекватна ли модель можно только в процессе экспериментов с ней, анализа результатов.

На рис. 7.2 проиллюстрированы две крайние ситуации: а) некоторый параметр x_i , очень сильно влияет на результирующую величину y_j , б) почти не влияет на нее. Ясно, что если все представляющие интерес величины y_j реагируют на x_i так, как изображено на рис. 7.2, б, то x_i является параметром, который при первом подходе может быть из модели исключен; если же хотя бы одна из величин y_j реагирует на изменение x_i так, как изображено на рис. 7.2, а, то x_i нельзя исключать из числа важнейших параметров.

Следующий этап - поиск математического описания. На этом этапе необходимо перейти от абстрактной формулировки модели к формулировке, имеющей конкретное математическое наполнение. В этот момент модель предстает перед нами в виде уравнения, системы уравнений, системы неравенств, дифференциального уравнения или системы таких уравнений и т.д.

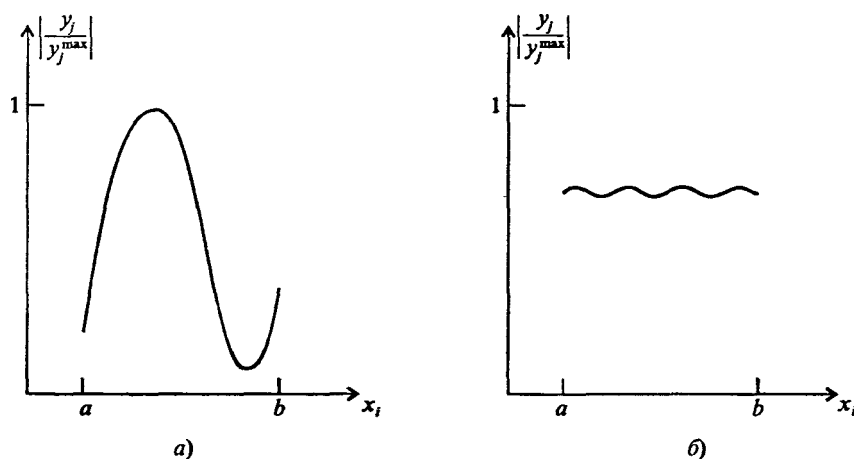


Рис. 7.2. Варианты степени влияния величины x_i на результирующую величину y_j

Когда математическая модель сформулирована, выбираем метод ее исследования. Как правило, для решения одной и той же задачи есть несколько конкретных методов, различающихся эффективностью, устойчивостью и т.д. От верного выбора метода часто зависит успех всего процесса.

Разработка алгоритма и составление программы для ЭВМ - это творческий и трудно формализуемый процесс. В настоящее время при компьютерном математическом моделировании наиболее распространенными являются приемы процедурно-ориентированного (структурного)

программирования, описанные в главе 3. Из языков программирования многие профессионалы-физики, например, до сих пор предпочитают FORTRAN как в силу традиций, так и в силу непревзойденной эффективности компиляторов (для расчетных работ) и наличия написанных на нем огромных, тщательно отлаженных и оптимизированных библиотек стандартных программ математической ориентации. В ходу и такие языки, как PASCAL, BASIC, C - в зависимости от характера задачи и склонностей программиста.

После составления программы решаем с ее помощью простейшую тестовую задачу (желательно, с заранее известным ответом) с целью устранения грубых ошибок. Это - лишь начало процедуры тестирования, которую трудно описать формально исчерпывающим образом. По существу, тестирование может продолжаться долго и закончиться тогда, когда пользователь по своим профессиональным признакам сочтет программу верной. Программистский фольклор полон историй об ошибках на этом пути.

Затем следует собственно численный эксперимент, и выясняется, соответствует ли модель реальному объекту (процессу). Модель адекватна реальному процессу, если некоторые характеристики процесса, полученные на ЭВМ, совпадают с экспериментальными с заданной степенью точности. В случае несоответствия модели реальному процессу возвращаемся к одному из предыдущих этапов.

2.3. КЛАССИФИКАЦИЯ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ

К классификации математических моделей разные авторы подходят по-своему, положив в основу классификации различные принципы. Можно классифицировать модели по отраслям наук (математические модели в физике, биологии, социологии и т.д.) - это естественно, если к этому подходит специалист в какой-то одной науке. Можно классифицировать по применяемому математическому аппарату (модели, основанные на применении обыкновенных дифференциальных уравнений, дифференциальных уравнений в частных производных, стохастических методов, дискретных алгебраических преобразований и т.д.) - это естественно для математика, занимающегося аппаратом математического моделирования. Наконец, человек, интересующийся общими закономерностями моделирования в разных науках безотносительно к математическому аппарату, ставящий на первое место цели моделирования, скорее всего заинтересуется такой классификацией:

- дескриптивные (описательные) модели;
- оптимизационные модели;
- многокритериальные модели;
- игровые модели;
- имитационные модели.

Остановимся на этом чуть подробнее и поясним на примерах. Моделируя движение кометы, вторгшейся в Солнечную систему, мы описываем (предсказываем) траекторию ее полета, расстояние, на котором она пройдет от Земли и т. д., т. е. ставим чисто описательные цели. У нас нет никаких возможностей повлиять на движение кометы, что-то изменить.

На другом уровне процессов мы можем воздействовать на них, пытаясь добиться какой-то цели. В этом случае в модель входит один или несколько параметров, доступных нашему влиянию. Например, меняя тепловой режим в зернохранилище, мы можем стремиться подобрать такой, чтобы достичь максимальной сохранности зерна, т. е. оптимизируем процесс.

Часто приходится оптимизировать процесс по нескольким параметрам сразу, причем цели могут быть весьма противоречивыми. Например, зная цены на продукты и потребность человека в пище, организовать питание больших групп людей (в армии, летнем лагере и др.) как можно полезнее и как можно дешевле. Ясно, что эти цели, вообще говоря, совсем не совпадают, т.е. при моделировании будет несколько критериев, между которыми надо искать баланс.

Игровые модели могут иметь отношение не только к детским играм (в том числе и компьютерным), но и к вещам весьма серьезным. Например, полководец перед сражением в условиях наличия неполной информации о противостоящей армии должен разработать план: в каком порядке вводить в бой те или иные части и т.д., учитывая и возможную реакцию противника. Есть специальный достаточно сложный раздел современной математики - теория игр, - изучающий методы принятия решений в условиях неполной информации.

Наконец, бывает, что модель в большой мере подражает реальному процессу, т.е. имитирует его. Например, моделируя изменение (динамику) численности микроорганизмов в колонии, можно рассматривать много отдельных объектов и следить за судьбой каждого из них, ставя определенные условия для его выживания, размножения и т.д. При этом иногда явное математическое описание процесса не используется, заменяясь некоторыми словесными условиями (например, по истечении некоторого отрезка времени микроорганизм делится на две части, а другого отрезка - погибает). Другой пример - моделирование движения молекул в газе, когда каждая молекула представляется в виде шарика, и задаются условия поведения этих шариков при столкновении друг с другом и со стенками (например, абсолютно упругий удар); при этом не нужно использовать никаких уравнений движения. Можно сказать, что чаще всего имитационное моделирование применяется в попытке описать свойства большой системы при условии, что поведение составляющих ее объектов очень просто и четко сформулировано. Математическое описание тогда производится на уровне статистической обработки результатов моделирования при нахождении макроскопических характеристик системы. Такой компьютерный эксперимент фактически претендует на воспроизведение натурального эксперимента: на вопрос «зачем же это делать» можно дать следующий ответ: имитационное моделирование позволяет выделить «в чистом виде» следствия гипотез, заложенных в наши представления о микрособытиях, очистив их от неизбежного в натурном эксперименте влияния других факторов, о которых мы можем даже не подозревать. Если же, как это иногда бывает, такое моделирование включает и элементы математического описания событий на микроуровне, и если исследователь при этом не ставит задачу поиска стратегии регулирования результатов (например, управления численностью колонии микроорганизмов), то отличие имитационной модели от дескриптивной достаточно условно; это, скорее, вопрос терминологии.

2.4. НЕКОТОРЫЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ

Как уже отмечалось выше, наибольшая польза при изучении вопросов, затрагиваемых в данной главе, будет в том случае, если, наряду с проникновением в общую методологию моделирования и осознанием существа рассматриваемых задач, читатели реализуют несколько моделей в виде компьютерных программ достаточно профессионального уровня, т. е. не просто заглатывающих одни числа и выводящих на экран или принтер длинные столбцы других, но и реализующих некоторый интерфейс пользователя с элементами диалога, графики и т.д.

Именно поэтому мы не будем касаться здесь вопроса об использовании специализированных пакетов программ решения математических задач и графической поддержки. Во-первых, такие пакеты различны на разных ЭВМ, постоянно появляются новые, и за этим не угнаться никакому учебнику. Во-вторых (и это главное), самостоятельно запрограммированное осознанное решение системы дифференциальных уравнений методом Эйлера и сопровождающая его простенькая самостоятельно созданная на BASIC или PASCAL иллюстрация в виде графика или движущегося по экрану предмета дают тому, кто это сделал, куда больше, чем обращение к пакету MATHEMATICA с его могучими программами. От того, что задача будет решена, скажем, программой из этого пакета по методу Рунге - Кутты - Мерсона с автоматическим выбором шага интегрирования, реальных знаний почти не прибавится. Здесь мы упираемся в фундаментальные проблемы целей образования. Разумеется, овладение возможностями одного из пакетов математической поддержки (Eureka, MathCad, MathLab, Derive и др.) вполне желательно и может принести определенную пользу, но не заменить самостоятельно проделанной работы.

Далее остановимся лишь на отдельных вопросах программирования, наиболее существенных при разработке моделирующих программ, ограничиваясь лишь примерами и советами, призванными уточнить суть дела.

Организация диалога человек - ЭВМ в процессе проектирования и разработки моделирующих программ. Организация интерфейса пользователя при разработке профессиональных программ - работа не менее (а часто и более) сложная, чем реализация логической или расчетной части задачи. Многие, работая с табличными процессорами, текстовыми редакторами и другими широко распространенными программами, не склонны задумываться, сколько усилий потребовала от разработчиков организация экрана, на котором всегда есть нужная в данный момент информация, причем именно в том месте экрана, где ее легче найти, система подсказок и помощи, меню для смены режимов работ и т.д. Очень жесткие требования по оформлению диалога предъявляют-

ся к обучающим и контролирующим Программам, компьютерным дидактическим играм, многим другим классам программ. Именно по соответствию этим требованиям сегодня в первую очередь судят о профессионализме разработки.

Характерной чертой практически всех профессиональных моделирующих программ является их диалоговый характер. Диалоговые (иначе говоря, интерактивные) программы - большой шаг вперед в развитии процесса взаимодействия человека с компьютером. Обсудим коротко лишь некоторые черты подобной организации человеко-машинного интерфейса.

Диалог человека с машиной весьма далек от привычного человеческого диалога. Главная особенность - ограниченные возможности компьютера в форме восприятия и анализа смысла человеческих сообщений. Воспринимаемые компьютером в ходе диалога сообщения жестко ограничены в своих конструкциях как синтаксически, так и семантически. Синтаксически - т. е. сообщения воспринимаются лишь в предусмотренной автором программы форме. Например, оговорено, что сообщение содержит 5 букв кириллицы - любое другое система признает неверным и отвергнет. Однако, она в этом случае сочтет верным любой текст из 5 букв кириллицы, даже совершенно бессмысленный; такой текст будет отвергнут на уровне семантического анализа, в котором исследуется, приписал ли автор программы этому тексту какой-либо смысл.

Слово «семантика» означает смысловую сторону сообщения. В распознавании смысла сообщений компьютерные программы пока преуспели гораздо меньше, чем в синтаксическом анализе, и могут несравненно меньше, чем человек. Работы в этом направлении - одно из самых перспективных направлений развития информатики. Подавляющее число программ, реализованных на компьютерах с первого по четвертое поколения, могут распознавать «верно - неверно» лишь по сравнению с эталонными ответами, которые либо заранее заложены в эти программы, либо выражены в числовой форме и рассчитываются программами.

Обсудим технические вопросы ведения диалога с компьютером. В моделирующих программах, в основном, используются следующие типы элементов диалога:

- меню;
- вопросы, требующие ответа «да/нет»;
- ответы по шаблону;
- команды.

Меню - на экране нумерованное или помеченное иным способом (например, выделением заглавных букв первых слов) множество функций системы, реализация которых возможна в настоящий момент, рис. 7. 3. Пользователь выбирает нужную функцию и тем самым начинает диалог.

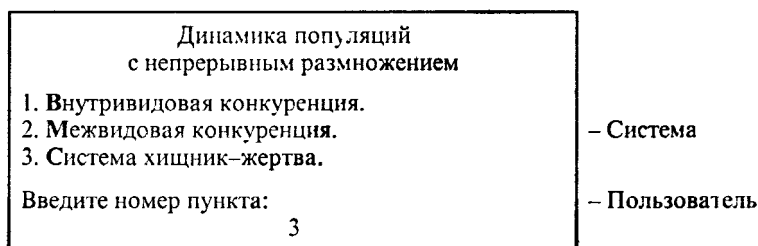


Рис. 7. 3, а. Пример кадра меню

Программа 146. Фрагмент программы, реализующей меню с рис. 7. 3,а.

```
Program St; Uses Crt;
Var M : Byte; Logic : Boolean;
(Про\"(o)e\"(a)ура печати меню и выбора соответствующего пункта)
Procedure Menu (Var M : Byte) ;
Begin Repeat (Выбор пункта меню с верифика\"(o)ией вво\"(a)a}
CtScr; GotoXY (15, 7);
Write('\"(A)ИНАМИКА ПОПУЛ\ss\"{0}ИЙ С НЕПРЕРЫВНЫМ РАЗМНОЖЕНИЕМ') •;
GotoXY(22, 10); Write('1. Внутриви\"{a}овая конкурен\"(o)ия. ');
GotoXY(22, 12); Write('2. Межви\"{a}овая конкурен\"(o)ия. ');
GotoXY(22, 14); Write('3. Система хищник-жертва. ');
GotoXY(22, 16); Write('4. Конеч\"{o} работы. ');
```

```

GotoXY(15, 22); Write('ВЫБЕРИТЕ НОМЕР НУЖНОГО ПУНКТА МЕНЮ ');
Read(M)
Until M In [1. . 4]
End;
Begin (ОСНОВНА\ss ПРОГРАММА)
  Logic := False;
  Repeat
    Menu(M); (меню)
    Case M Of
      1 : Begin (Внутриви\"{a}овая конкурен\"{o}ия) End;
      2 : Begin {Межви\"{a}овая конкурен\"{o}ия)
        CIrScr; WriteLn('Пункт меню "Межви\"{a}овая конкурен\"{o}ия"') ;
        Repeat Until KeyPressed
      End;
      3 : Begin {Система хищник-жертва} End;
      4 : Logic := True (Выхо\"{a} из программы)
    End
  Until Logic
End.

```

Программа может предусматривать и другие способы организация реакции пользователя: вводом букв В, М, С; еще удобнее - установкой курсора в нужную строку клавишами и (или) «мышью» (если она есть). Суть дела это не меняет.

Вопрос, требующий ответа «да/нет» - частный случай меню, который, однако, целесообразно выделить.

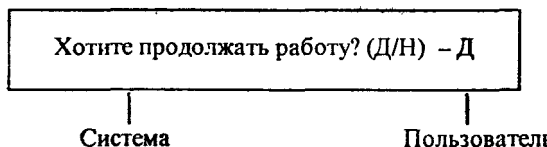


Рис. 7. 3, б. Выбор одного из возможных ответов

Ответы по шаблону даются чаще всего при ожидании числовой информации, когда хотят избежать длинных объяснений о формате числа, количестве значащих цифр.

Команда подается пользователем. В диалоговой моделирующей программе может быть строго фиксированный набор команд, краткий перечень которых сопровождает все или часть кадров в строке-подсказке; по этим командам пользователь может в любой момент прекратить работу или изменить ее характер. Например, в нижней части кадра, содержание которого сейчас неважно, написано:

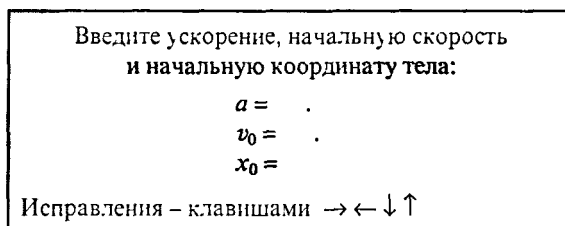


Рис. 7. 3, в. Форматированный ввод данных

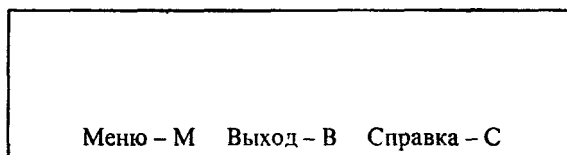


Рис. 7, 3, г. Выбор последующего режима работы

В основной части кадра-некий текст, предписывающий определенную реакцию. Но указанные внизу команды имеют приоритет, и при нажатии «М» мы попадем в основное меню незави-

симо от того, что требовалось в верхней части кадра. При этом предусматривается возможность возврата в кадр, из которого вышли.

Разработка диалоговой моделирующей программы - достаточно сложный процесс, включающий создание сценария, программирование, разработка инструкций и документации. В нашем курсе невозможно требовать реализации всех этих элементов в каждой из выполняемых работ, поскольку тогда программистская работа отодвинет на второй план все остальное; можно лишь рекомендовать использовать элементы организации диалога. В отчетной (курсовой, дипломной) работе, напротив, уместна реализация весьма интересных диалоговых оболочек; в текущих же работах элементы разумной организации диалога, оформления экрана вполне уместны и не требуют чрезмерных усилий. Весь диалог при моделировании того уровня, который соответствует данному курсу, невелик по объему и несложен по логической организации. Согласитесь, что ситуация, когда программа требует ввода данных и извещает об этом лишь знаком «?» на совершенно пустом экране, вам, вероятно, знакома, как знакома и реакция сидящего за компьютером человека, который не понимает, что бы это значило (что особенно смешно, если он сам - автор этой программы). Так не стоит ли снабдить этот знак фразой «введите начальную скорость v_0 , звуковым сигналом, привлекающим внимание, и т.д. ? Как быть с огромным количеством чисел, проносящихся по экрану в процессе решения дифференциальных уравнений и сметающих на своем пути всю нужную информацию? В отлаженной программе такого просто не должно быть.

Спроектируем возможное оформление диалога и формы представления результатов для одной из рассмотренных ниже задач по моделированию. Уместно начать с кадра-заставки, где написаны название задачи, фамилия автора программы - это минимум. Если удастся создать несложный рисунок, иллюстрирующий задачу - очень хорошо, он уместен в том же кадре. Затем, через несколько секунд, кадр сменяется следующим. На нем - основное уравнение (т.е. сама математическая модель) и предложение ввести исходные данные - с представлением шаблонов, или, если это затруднительно, то в свободной форме, но обязательно со словесным запросом значения каждого параметра по очереди. Более чем существенно разумно спроектировать формы для вывода результатов - таблицы, графики, траектории и т.д., их очередность, взаимное расположение.

Неплохо, если в процессе счета программа выводит на экран стационарную или мерцающую надпись типа «Подождите, идут расчеты». Когда счет закончен, на экране может появиться меню с предложениями: «Вывод таблицы результатов», «Вывод графика $N(t)$ », «Запись параметров и результатов в файл», «Печать результатов», «Выполнение очередного расчета», «Окончание расчетов».

Разумеется, можно усовершенствовать это оформление, но каждый шаг здесь требует определенной программистской культуры и немалого труда.

Повсеместный переход на графические интерфейсные операционные системы уровня WINDOWS, появление сред визуального программирования типа DELPHI ведут к значительному повышению стандартизации пользовательского интерфейса. Если при разработке моделирующих программ читатели используют возможности таких сред, то могут получить вполне профессиональные программные продукты.

Отметим, наконец, что те несколько программ, которые приведены ниже при рассмотрении конкретных моделей, не претендуют на совершенный интерфейс. Их цель - пояснение существа рассматриваемых процессов; попытка же оформить их «по всем правилам» привела бы к большому, плохо читаемым программам, не выполняющим этой разъяснительной функции.

Компьютерная научная графика. С простейшей научной графикой мы встречаемся очень рано. Уже в курсе математики 6-7 классов есть достаточно абстрактные и условные рисунки, которые дети легко воспринимают - например, график линейной функции. А ведь на нем немало элементов, интерпретация которых, если задуматься, вовсе не очевидна: линии, штрихи, стрелки, масштабы и т.д. Несмотря на это, понять по графику свойства сложной функции человеку гораздо легче, чем из соответствующей формулы, хотя в ней информации, строго говоря, гораздо больше. Так уж устроено человеческое восприятие, что рисунки, пусть даже условные, гораздо легче воспринимаются рассудком, чем сложные формулы или колонки чисел.

В современной прикладной информатике этим обстоятельством очень широко пользуются, и в ней сформировалось соответствующее направление - машинная (компьютерная) графика. По определению, машинная графика - раздел информатики, в рамках которого исследуются и разрабатываются технические, математические, программные и методические средства и приемы ис-

пользования ЭВМ для создания, обработки, хранения и практического применения графических изображений.

В машинной графике выделяют несколько разделов.

Иллюстративная графика, простейшими программными средствами которой являются всем знакомые диалоговые программы - графические редакторы, служит для создания изображений, за которыми, как правило, не стоят какие-либо математические объекты (уравнения и др.). Это - средство реализации свободного полета мысли и воображения, любимое занятие начинающих приобщаться к компьютеру.

Деловая графика существенно «скучнее». Когда администратору, бухгалтеру, экономисту и т.д. нужно перевести сухие колонки чисел в столбчатую диаграмму, круговую диаграмму, график, достаточно вызвать такую программу и в ходе диалога сообщить ей заголовки, подписи, разметки, цвета и т.д. и имя файла, в котором по определенным правилам записаны указанные числа. Система построит заданное изображение на экране, выведет его на принтер.

Одна из самых сложных и специализированных разновидностей систем машинной графики - *инженерная графика*, известная также под именем САПР - системы автоматизированного проектирования. Это диалоговые системы, предназначенные для автоматизации процесса проектирования технических объектов, создания полных комплектов проектных документов с учетом существующих норм стандартов.

И, наконец, *научная графика* - наиболее актуальная для изучаемого курса и наименее всех допускающая единое описание. Универсальных систем компьютерной научной графики, по-видимому, не существует из-за огромного разнообразия задач. Часто программы, реализующие наглядное изображение решения научной задачи (почти всегда по итогам математического моделирования), встраиваются внутрь основной программы, пишутся на том же самом языке программирования.

Общую цель научной графики можно сформулировать так: сделать невидимое и абстрактное «видимым». Берем последнее слово в кавычки, так как часто эта «видимость» весьма условна. Можно ли увидеть распределение температуры внутри неоднородно нагретого тела сложной формы без введения в него сотен микродатчиков, т.е. , по существу, его разрушения? - Да, если есть соответствующая математическая модель, и, что очень важно - договоренность о восприятии определенных условностей на рисунке. Можно ли увидеть распределение металлических руд под землей без раскопок? Строение поверхности чужой планеты по результатам радиолокации? На эти и множество других вопросов ответ - да, можно, с помощью машинной графики и предшествующей ей математической обработки. Изображения такого рода систематически публикуются научными и научно-популярными изданиями.

Более того, можно «увидеть» и то, что строго говоря, вообще плохо соответствует слову «видеть». Так, возникшая на стыке химии и физики наука - квантовая химия - дает нам возможность «увидеть» строение молекулы. Эти изображения - верх абстракции и системы условностей, так как в атомном мире обычные наши понятия о частицах (ядрах, электронах и т.д.) принципиально не применимы. Однако, многоцветное «изображение» молекулы на экране компьютера для тех, кто понимает всю меру его условности, приносит большую пользу, чем тысячи чисел, являющихся плодом квантовохимического расчета.

При реализации относительно несложных задач нашего курса прибегать к помощи каких-либо стандартных пакетов машинной графики - дело вовсе необязательное. Конечно, можно себе представить, что итоговое оформление экрана в конце процесса моделирования выполнено с помощью графического редактора, а содержащийся в нем график - с помощью пакета деловой графики (или еще какого-то), но возникающую при этом проблему совмещения разных систем программирования не всегда легко решить. Целесообразнее, по-видимому, ориентироваться на тот язык программирования, на котором реализуется математическая модель.

Приведем несколько конкретных примеров, привязанных к нашему курсу.

Траектории движения тел, графики. В ряде рассмотренных ниже задач уместно иллюстрировать процесс моделирования изображениями движущихся объектов и их траекториями. Мы сознательно ограничивались случаями плоских (двумерных) движений, которые легко отобразить на плоском экране компьютера.

Поскольку основные графические операторы языка BASIC или процедуры модуля GRAPH Turbo Pascal нашим читателям известны, опишем лишь общие моменты построения графиков и

траекторий. Напомним, что в главе 3 в разделе, посвященном PASCAL, приведена программа построения графика аналитически заданной функции на произвольном отрезке; здесь мы детализируем рассмотрение. Пусть численные расчеты уже закончены и нам известны границы значений координат $[x_{\min}, x_{\max}]$ и $[y_{\min}, y_{\max}]$ и есть таблица значений x и y в некоторые моменты времени, разделенные равными промежутками: $0, t, 2t, 3t, \dots, nt$. Требуется построить графики зависимости $x(t), y(t)$ и траекторию. Проиллюстрируем это, используя графические процедуры PASCAL.

С помощью директивы Uses Graph и процедуры InitGraph (<параметры>) осуществляется переход в графический режим, в котором можно строить изображения. Необычная ориентация «экранной» системы координат создает определенные проблемы при построении графиков и траекторий. Мы хотим выводить их и задавать координаты точек в «естественной» системе координат x, y , изображенной на рис. 7.4, а графические процедуры (Circle, Line, OutText и др.) воспринимают аргументы в «экранной» системе x', y' . Сделаем разметку так, как показано на рисунке, и произведем линейное преобразование координат

$$x' = \alpha x + \beta, \quad y' = \gamma y + \delta.$$

Если известны разрешающая способность экрана - M точек по оси x' и N точек по оси y' , то для нахождения коэффициентов $\alpha, \beta, \gamma, \delta$ достаточно связать любые две точки в разных системах координат, например

$$(x = x_{\min}, y = y_{\min}) \Rightarrow (x' = 10, y' = N - 10),$$

$$(x = x_{\max}, y = y_{\max}) \Rightarrow (x' = M - 10, y' = 10)$$

(отступ на 10 позиций от краев экрана позволит создавать подписи, разметку осей и др.). Имеем

$$\begin{cases} 10 = \alpha \cdot x_{\min} + \beta, \\ M - 10 = \alpha \cdot x_{\max} + \beta, \end{cases}$$

откуда

$$\begin{aligned} \alpha &= \frac{M - 20}{x_{\max} - x_{\min}}, \\ \beta &= \frac{10x_{\max} - (M - 10)x_{\min}}{x_{\max} - x_{\min}}, \\ \begin{cases} N - 10 = \gamma \cdot y_{\min} + \delta, \\ 10 = \gamma \cdot y_{\max} + \delta, \end{cases} \end{aligned}$$

откуда

$$\begin{aligned} \gamma &= \frac{20 - N}{y_{\max} - y_{\min}}, \\ \delta &= \frac{(N - 10)y_{\max} - 10y_{\min}}{y_{\max} - y_{\min}}. \end{aligned}$$

Таким образом, перевод одних координат в другие осуществляется по формулам

$$\begin{aligned} x' &= \frac{(M - 20)x + 10x_{\max} - (M - 10)x_{\min}}{x_{\max} - x_{\min}}, \\ y' &= \frac{(20 - N)y + (N - 10)y_{\max} - 10y_{\min}}{y_{\max} - y_{\min}}. \end{aligned}$$

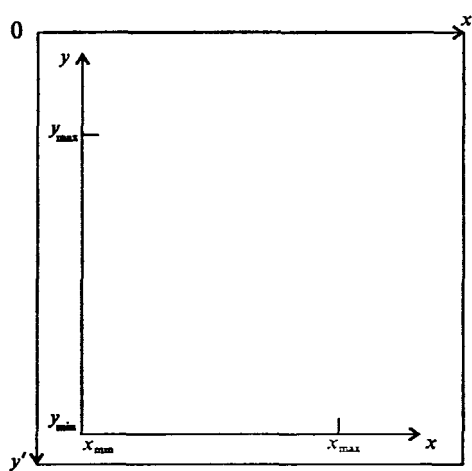


Рис. 7.4. Экранная и «естественная» системы координат

Теперь достаточно поставить точку с нужной координатой (x, y) с помощью процедуры PutPixel, а введя ее в цикл, изобразить график или траекторию. Если же требуется изобразить движение тела, то перед выводом на экран очередной точки достаточно стереть предыдущую или воспользоваться несколькими видеостраницами - соответствующие приемы программирования читателю, скорее всего, известны. Отметим, что создание на экране дисплея динамических зрительных изображений - так называемая анимация - одно из перспективных направлений искусственного интеллекта-раздела современной информатики.

Изолинии. В задачах моделирования достаточно стандартная проблема - построение линий (поверхностей), вдоль которых некоторая функция имеет одинаковое значение, называемых изолиниями (изоповерхностями). Это очень распространенная задача визуализации характеристик некоторого скалярного поля в приближении сплошной среды: изотермы - линии равной температуры, изобары - линии равного давления, изолинии функции тока жидкости или газа, по которым легко можно представить себе их потоки, изолинии численностей экологической популяции на местности, изолинии концентрации вредных примесей в окружающей среде и т.д.

Опишем типичную процедуру построения изолиний на экране компьютера. На старте мы имеем двумерную таблицу значений некоторой величины A , полученную в ходе математического моделирования; числа в этой таблице соответствуют значениям этой величины в узлах пространственной сетки (рис. 7.5).

Зададим некоторый, совершенно условный, пространственный шаг h между соседними узлами по горизонтали и вспомогательную систему координат, в которой узел $(1, 1)$ имеет координату $(0, 0)$, узел $(1, 2)$ - координату $(h, 0)$, узел $(1, 3)$ - координату $(2h, 0)$ и т.д. Если шаг по вертикали h^* , то узел (i, j) в этой системе имеет координату $((i-1) \cdot h, (j-1) \cdot h^*)$.

Предварительно найдем в таблице наибольшее и наименьшее значения величин a_{ij} - допустим, это $a_{\min}$ и $a_{\max}$. Пусть b - некоторое промежуточное значение: $a_{\min} < b < a_{\max}$. Обсудим в общих чертах, как построить изолинию $A = b$. Будем для этого (в цикле) просматривать вначале все пары ближайших чисел в первой строке таблицы в поисках такой пары, для которой b находится «внутри». Допустим, число b находится между a_{1k} и $a_{1,k+1}$, т.е. либо $a_{1k} < b < a_{1,k+1}$, либо $a_{1k} > b > a_{1,k+1}$.

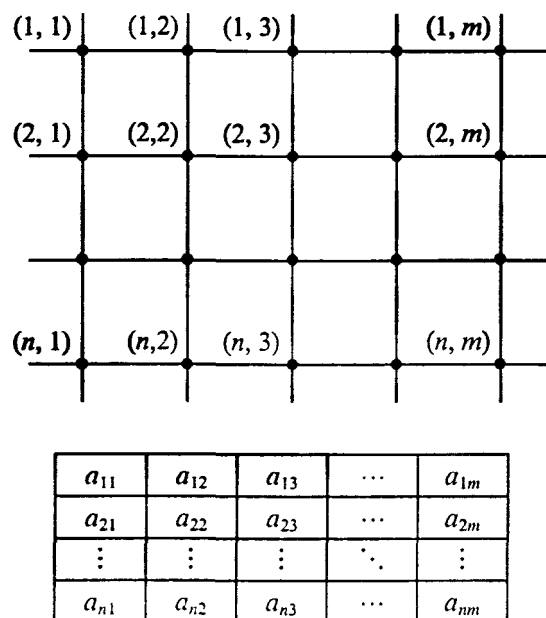


Рис. 7.5. Пространственная сетка и соответствующая ей таблица значений величины A

С помощью линейной интерполяции найдем соответствующую горизонтальную координату точки, в которой $A = b$:

$$x = (k-1) \cdot h + \frac{b - a_{1k}}{a_{1,k+1} - a_{1k}} h$$

(координата y определяется номером горизонтальной линии; в данном случае $y = 0$).

Найденные координаты запомним и просмотрим первую строку в таблице до конца, затем просмотрим вторую строку и т.д. Покончив с просмотром строк, мы получим часть точек, соответствующих изолинии $A = b$.

После этого займемся просмотром столбцов. Допустим, во втором столбце нашлась пара чисел, для которой число b находится между a_{p2} и $a_{p+1,2}$. Она дает следующую точку для изолинии. Закончив просмотр всех столбцов, мы получим максимально возможный набор координат точек, принадлежащих данной изолинии. Выведя их на экран в нужном масштабе, получим точечное изображение изолинии $A = b$, после чего можем, взяв другое значение b , построить следующую изолинию. Более детально эта процедура изложена ниже в пункте 3.8 на примере построения линий равного потенциала электрического поля.

Условные цвета, условное контрастирование. Еще один интересный прием современной научной графики - условная раскраска. Она находит широчайшее применение в самых разных приложениях науки и представляет собой набор приемов по максимально удобной, хотя и очень условной, визуализации результатов компьютерного моделирования.

Приведем примеры. В различных исследованиях температурных полей встает проблема наглядного представления результатов. Самый простой (и, с точки зрения специалиста, весьма неэффективный) - привести карту (чертеж, план), в некоторых точках которой обозначены значения температуры.

Другой способ - набор изотерм - гораздо эффективнее; к нему прибегают некоторые газеты, давая состояние и прогноз погоды. Но можно добиться еще большей наглядности, учитывая, что большинству людей свойственно, сравнивая разные цвета, воспринимать красный как «горячий», голубой как «холодный», а все остальные - между ними. Допустим, что на некоторой территории температура в данный момент имеет в разных местах значения от -25°C до $+15^{\circ}\text{C}$. Разделим этот диапазон на участки с шагом, равным, например, 5°

$[-25, -20], [-20, -15], \dots, [+10, +15],$

и закрасим первый из них в ярко-голубой, последний - в ярко-красный, а все остальные - в проме-

жуточные оттенки голубого и красного цветов. Получится замечательная наглядная картина температурного поля.

А что делать, если дисплей монохромный?! Или если изображение надо перенести с цветного дисплея на бумагу при отсутствии возможности цветной печати? -Тогда роль цвета может сыграть контраст. Сделаем самый «горячий» участок самым темным, самый «холодный» - прозрачным, а остальные - между ними. Эффектность, конечно, меньше, чем при цветовой раскраске, но для наметанного глаза изображение информативно.

То же самое можно делать при иллюстрации температурного поля и на поверхности обрабатываемой на станке детали, и на поверхности далекой планеты.

В нашем курсе есть несколько моделей, в которых можно (и очень полезно) прибегнуть к подобному приему визуализации. В задаче о теплопроводности в стержне это даже не очень сложно; можно делать такие условные раскраски при моделировании распределения электрических полей. Если заниматься имитационным моделированием конкурирующих популяций, то, раскрасив их в разные цвета, можно получить на экране причудливые картины, передающие ход конкурентной борьбы.

Условные раскраски бывают и гораздо более абстрактными, чем в описанных выше случаях. При моделировании сложных органических молекул компьютер может выдавать результаты в виде многоцветной картины, на которой атомы водорода изображены одним цветом, углерода - другим и т.д., причем атом представлен шариком (кружочком), в пределах которого плотность цвета меняется в соответствии с распределением электронной плотности.

При поиске полезных ископаемых методами аэрофотосъемки с самолетов или космических спутников компьютеры строят условные цветовые изображения распределений плотности под поверхностью Земли. Подобных примеров можно привести достаточно много.

Подведем итог: изображения в условных цветах и контрастах - мощнейший прием научной графики. Он позволяет понять строение не только плоских, но и объемных (трехмерных) объектов, дает в руки исследователя один из замечательных методов познания. Приведем в качестве иллюстрации фрагмент программы.

Программа 147. Условная раскраска неравномерно нагретого стержня в разные моменты времени (по заранее заготовленным данным).

```
Program Stergen;
Uses Crt, Graph
Type Mas2 = Array [0..10, 0..4] of Real;
Const (Массив распределения температуры в разные моменты времени)
U : Mas2 =
((3.000, 3.667, 4.333, 5.000, 3.000), (3.000, 3.628, 4.128, 3.952, 3.000),
 (3.000, 3.514, 3.783, 3.593, 3.000), (3.000, 3.377, 3.546, 3.396, 3.000),
 (3.000, 3.267, 3.381, 3.272, 3.000), (3.000, 3.187, 3.266, 3.188, 3.000),
 (3.000, 3.131, 3.185, 3.131, 3.000), (3.000, 3.091, 3.129, 3.091, 3.000),
 (3.000, 3.064, 3.090, 3.064, 3.000), (3.000, 3.044, 3.063, 3.044, 3.000),
 (3.000, 3.031, 3.044, 3.031, 3.000));
Var
M, I, J, N1, Nt : Integer; MaxF, L, T, HI, Ht : Real;
Procedure Initialize; (Процедура инициализации графического режима)
Var GraphDriver, GraphMode : Integer;
Begin
DetectGraph(GraphDriver, GraphMode) ;
InitGraph(GraphDriver, GraphMode, '');
End;
(Графическая иллюстрация решения)
Procedure Postar.ovka (U : Mas2; Nt, N1 : Integer; HI, L, MaxF : Real);
Var X_N, Shag, Y_N, Shir, Dlin, Color, I, J, K, Y : Integer;
Flag : Boolean; Ff : String; Col : Array [0..15] Of Byte;
Begin
Initialize; (Инициализация графического режима)
X_N := GetMaxX Div 6;
If Nt <= 6 Then M := Nt Else M := Nt Div 2;
Y_N := GetMaxY Div M - 20; Shir := Y_N Div 2;
```

```

Dlin := GetMaxX -2 * X_N; Shag := Trunc(Dlin / N1); Str(Shag,Ff) ;
Col[0] := 0; Col[1] := 8; Col[2] := 1; (Палитра цветов)
Col[3] := 9; Col[4] := 3; Col[5] := 11;
Col[6] := 2; Col[7] := 10; Col[8] := 14;
Col[9] := 13; Col[10] := 5; Col[11] := 12; Col [12] := 4;
For I := 0 To M - 1 Do (номер временного промежутка)
  Begin
    For J := 0 To N1 - 1 Do (номер участка стержня)
      Begin
        Flag := False;
        For K := 0 To Shag Do
          Begin
            For y:= 0 To Shir Do
              Begin
                Color := 1 + Round((U[I, J] + (определение номера цвета)
                (U[I, J + 1] - U[I, J]) * K / Shag - U[0, 0]) / 3 * 16);
                If Random(64) > 32
                Then If Random(64) > 32 Then Color := Color + 1 Else
                  Color := Color - 1;
                If Not Flag Then (вывод текущей температуры)
                Begin
                  Str((U[I,J]+(U(I,J+1)-U[I,J])*K/Shag) : 5 : 3, Ff);
                  OutTextXY(K+X_N+Shag*J, Y_N*(1+1)-19, Ff) ;
                  Flag := True
                End;
                {рисование точки}
                PutPixel(K+X_N+Shag*J, Y+Y_N*(1+I), Col[Color])
              End
            End
          End
        End;
      End;
    End;
  End;
  SetColor(White); OutTextXY(150, 450, 'Нажмите любую клавишу ');
  Repeat Until KeyPressed; CloseGraph
End;
Begin (ОСНОВНАЯ ПРОГРАММА)
  L := 4; T := 10; Hi := 1; Ht := 1;
  N1 := Trunc(L / HI); Nt := Trunc(T / Ht); MaxF := 5;
  Postanovka (U. Nt, N1, HI, L, MaxF)
End.

```

§3. МОДЕЛИРОВАНИЕ ФИЗИЧЕСКИХ ПРОЦЕССОВ

3.1. ФИЗИКА И МОДЕЛИРОВАНИЕ

Физика - наука, в которой математическое моделирование является чрезвычайно важным методом исследования. Наряду с традиционным делением физики на экспериментальную и теоретическую сегодня уверенно выделяется третий фундаментальный раздел - вычислительная физика (*computational physics*). Причину этого в целом можно сформулировать так: при максимальном проникновении в физику математических методов, порой доходящем до фактического сращивания этих наук, реальные возможности решения возникающих математических задач традиционными методами очень ограничены. Из многих конкретных причин выделим две наиболее часто встречающиеся: нелинейность многих физических процессов (примеры - ниже в тексте) и необходимость исследования совместного движения многих тел, для которого приходится решать системы большого числа уравнений. Часто численное моделирование в физике называют вычислительным экспериментом, поскольку оно имеет много общего с лабораторным экспериментом.

Таблица 7.1

Аналогии между лабораторным и вычислительным экспериментами

Лабораторный эксперимент	Вычислительный эксперимент
Образец	Модель
Физический прибор	Программа для компьютера
Калибровка прибора	Тестирование программы
Измерение .	Расчет
Анализ данных	Анализ данных

Численное моделирование (как и лабораторные эксперименты) чаще всего является инструментом познания качественных закономерностей природы. Важнейшим его этапом, когда расчеты уже завершены, является осознание результатов, представление их в максимально наглядной и удобной для восприятия форме. Забить числами экран компьютера или получить распечатку тех же чисел не означает закончить моделирование (даже если числа эти верны). Тут на помощь приходит другая замечательная особенность компьютера, дополняющая способность к быстрому счету - возможность визуализации абстракций. Представление результатов в виде графиков, диаграмм, траекторий движения динамических объектов в силу особенностей человеческого восприятия обогащает исследователя качественной информацией. Во многих рассматриваемых ниже физических задачах фундаментальную роль играет второй закон Ньютона - основа всей динамики:

$$\vec{a} = \frac{\vec{F}}{m}.$$

В уточненной редакции закон утверждает: ускорение, с которым движется тело в данный момент времени, пропорционально действующей на него в этот момент силе и обратно пропорционально имеющейся в данный момент у тела массе.

Разные записи этого утверждения:

$$\vec{a}(t) = \frac{\vec{F}(t)}{m(t)}, \quad \frac{d\vec{v}}{dt} = \frac{\vec{F}(t)}{m(t)}, \quad \frac{d^2\vec{S}}{dt^2} = \frac{\vec{F}(t)}{m(t)}.$$

Связывая мгновенные значения величин, второй закон Ньютона позволяет изучать движение тел при произвольных изменениях во времени силы и массы.

3.2. СВОБОДНОЕ ПАДЕНИЕ ТЕЛА С УЧЕТОМ СОПРОТИВЛЕНИЯ СРЕДЫ

При реальных физических движениях тел в газовой или жидкостной среде трение накладывает огромный отпечаток на характер движения. Каждый понимает, что предмет, сброшенный с большой высоты (например, парашютист, прыгнувший с самолета), вовсе не движется равноускоренно, так как по мере набора скорости возрастает сила сопротивления среды. Даже эту, относительно несложную, задачу нельзя решить средствами «школьной» физики; таких задач, представляющих практический интерес, очень много. Прежде чем приступить к обсуждению соответствующих моделей, вспомним, что известно о силе сопротивления.

Закономерности, обсуждаемые ниже, носят эмпирический характер и отнюдь не имеют столь строгой и четкой формулировки, как второй закон Ньютона. О силе сопротивления среды движущемуся телу известно, что она, вообще говоря, растет с ростом скорости (хотя это утверждение не является абсолютным). При относительно малых скоростях величина силы сопротивления пропорциональна скорости и имеет место соотношение $F_{\text{сопр}} = k_1 v$, где k_1 определяется свойствами среды и формой тела. Например, для шарика $k_1 = 6\pi\mu r$ - это формула Стокса, где μ - динамическая вязкость среды, r - радиус шарика. Так, для воздуха при $t = 20^\circ\text{C}$ и давлении 1 атм. $\mu = 0,0182 \text{ Н}\cdot\text{с}\cdot\text{м}^{-2}$, для воды $1,002 \text{ Н}\cdot\text{с}\cdot\text{м}^{-2}$, для глицерина $1480 \text{ Н}\cdot\text{с}\cdot\text{м}^{-2}$.

Оценим, при какой скорости для падающего вертикально шара сила сопротивления сравняется с силой тяжести (и движение станет равномерным).

Имеем

$$6\pi\mu r v^* = mg,$$

или

$$v^* = \frac{mg}{6\pi\mu r} = \frac{(4/3)\pi r^3 \rho g}{6\pi\mu r} = \frac{2r^2 \rho g}{9\mu}.$$

Пусть $r = 0,1$ м, $\rho = 0,8 \cdot 10^3$ кг/м³ (дерево). При падении в воздухе $v^* \approx 960$ м/с, в воде $v^* \approx 17$ м/с, в глицерине $v^* \approx 0,012$ м/с.

На самом деле первые два результата совершенно не соответствуют действительности. Дело в том, что уже при гораздо меньших скоростях сила сопротивления становится пропорциональной квадрату скорости: $F_{\text{сопр}} = k_2 v^2$. Разумеется, линейная по скорости часть силы сопротивления формально также сохранится, но если $k_2 v^2 \gg k_1 v$, то вкладом $k_1 v$ можно пренебречь (это конкретный пример ранжирования факторов). О величине k_2 известно следующее: она пропорциональна площади сечения тела S , поперечного по отношению к потоку, и плотности среды $\rho_{\text{среды}}$ и зависит от формы тела. Обычно представляют $k_2 = 0,5 c S \rho_{\text{среды}}$, где c - коэффициент лобового сопротивления - безразмерен. Некоторые значения c (для не очень больших скоростей) приведены на рис. 7.6.

При достижении достаточно большой скорости, когда образующиеся за обтекаемым телом вихри газа или жидкости начинают интенсивно отрываться от тела, значение c в несколько раз уменьшается; для шара оно становится приблизительно равным 0,1. Подробности можно найти в специальной литературе.

Вернемся к указанной выше оценке, исходя из квадратичной зависимости силы сопротивления от скорости.

Имеем

$$cS \frac{\rho}{2} v^{*2} = mg$$

или

$$v^* = \sqrt{\frac{2mg}{cS\rho}}; \quad (7.4)$$

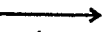
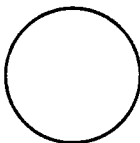
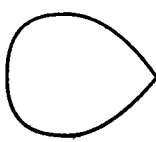
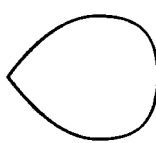
		Диск	$c = 1,11$
		Полусфера	$c = 1,33$
		Полусфера	$c = 0,55$
		Шар	$c = 0,4$
		Каплевидное тело	$c = 0,045$
		Каплевидное тело	$c = 0,1$

Рис. 7.6. Значения коэффициента лобового сопротивления для некоторых тел, поперечное сечение которых имеет указанную на рисунке форму (см. книгу П.А.Стрелкова)

Для шарика

$$v^* = \sqrt{\frac{2\rho_{\text{тела}} \cdot (4/3)\pi r^3 g}{c\pi r^2 \rho_{\text{среды}}}}; \quad (7.5)$$

Примем $r = 0,1$ м, $\rho = 0,8 \cdot 10^3$ кг/м³ (дерево). Тогда для движения в воздухе ($\rho_{\text{возд}} = 1,29$ кг/м³) получаем $v^* \approx 18$ м/с, в воде ($\rho_{\text{воды}} \approx 1 \cdot 10^3$ кг/м³) $v^* \approx 0,65$ м/с, в глицерине ($\rho_{\text{глицерина}} = 1,26 \cdot 10^3$ кг/м³) $v^* \approx 0,58$ м/с.

Сравнивая с приведенными выше оценками линейной части силы сопротивления, видим, что для движения в воздухе и в воде ее квадратичная часть сделает движение равномерным задолго до того, как это могла бы сделать линейная часть, а для очень вязкого глицерина справедливо обратное утверждение. Рассмотрим свободное падение с учетом сопротивления среды. Математическая модель движения - уравнение второго закона Ньютона с учетом двух сил, действующих на тело; силы тяжести и силы сопротивления среды:

$$\frac{d\vec{v}}{dt} = \frac{\vec{F}_{\text{тяж}} + \vec{F}_{\text{сопр}}}{m}. \quad (7.6)$$

Движение является одномерным; проецируя векторное уравнение на ось, направленную вертикально вниз, получаем

$$\frac{dv}{dt} = \frac{mg - k_1 v - k_2 v^2}{m}. \quad (7.7)$$

Вопрос, который мы будем обсуждать на первом этапе, таков: каков характер изменения скорости со временем, если все параметры, входящие в уравнение (7.7), заданы? При такой постановке модель носит сугубо дескриптивный характер. Из соображений здравого смысла ясно, что при наличии сопротивления, растущего со скоростью, в какой-то момент сила сопротивления сравняется с силой тяжести, после чего скорость больше возрастать не будет. Начиная с этого момента, $dv/dt = 0$, и соответствующую установившуюся скорость $\tilde{v}$ можно найти из условия $mg - k_1 v - k_2 v^2 = 0$, решая не дифференциальное, а квадратное уравнение. Имеем

$$\tilde{v} = \sqrt{\frac{k_1^2}{4k_2^2} + \frac{mg}{k_2}} - \frac{k_1}{2k_2} \quad (7.8)$$

(второй - отрицательный - корень, естественно, отбрасываем). Итак, характер движения качественно таков: скорость при падении возрастает от v_0 до $\tilde{v}$; как и по какому закону - это можно узнать, лишь решив дифференциальное уравнение (7.7).

Однако, даже в столь простой задаче мы пришли к дифференциальному уравнению, которое не относится ни к одному из стандартных типов, выделяемых в учебниках по дифференциальным уравнениям, допускающих очевидным образом аналитическое решение. И хотя это не доказывает невозможность его аналитического решения путем хитроумных подстановок, но они не очевидны (один из лучших помощников в их поиске - справочник Камке). Допустим, однако, что нам удастся найти такое решение, выраженное через суперпозицию нескольких алгебраических и трансцендентных функций - а как найти закон изменения во времени перемещения? - Формальный ответ прост:

$$S(t) = \int_{t_0}^t v(\tau) d\tau, \quad (7.9)$$

но шансы на реализацию этой квадратуры уже совсем невелики. Дело в том, что класс привычных нам элементарных функций очень узок, и совершенно стандартна ситуация, когда интеграл от суперпозиции элементарных функций не может быть выражен через элементарные функции в принципе. Математики давно расширили множество функций, с которыми можно работать почти так же просто, как с элементарными (т.е. находить значения, различные асимптотики, строить графики, дифференцировать, интегрировать). Тем, кто знаком с функциями Бесселя, Лежандра, интегральными функциями и еще двумя десятками других, так называемых, специальных функций, легче находить аналитические решения задач моделирования, опирающихся на аппарат дифференциальных уравнений. Однако даже получение результата в виде формулы не снимает проблемы представления его в виде, максимально доступном для понимания, чувственного восприятия, ибо мало кто может, имея формулу, в которой сопряжены логарифмы, степени, корни, синусы и тем более специальные функции, детально представить себе описываемый ею процесс - а именно это есть цель моделирования.

В достижении этой цели компьютер - незаменимый помощник. Независимо от того, какой будет процедура получения решения - аналитической или численной, - задумаемся об удобных способах представления результатов. Разумеется, колонки чисел, которых проще всего добиться от компьютера (что при табулировании формулы, найденной аналитически, что в результате численного решения дифференциального уравнения), необходимы; следует лишь решить, в какой форме и размерах они удобны для восприятия. Слишком много чисел в колонке быть не должно,

их трудно будет воспринимать, поэтому шаг, с которым заполняется таблица, вообще говоря, гораздо больше шага, с которым решается дифференциальное уравнение в случае численного интегрирования, т.е. далеко не все значения v и S , найденные компьютером, следует записывать в результирующую таблицу (табл. 7.2).

Таблица 7.2

Зависимость перемещения и скорости падения «безпарашютиста» от времени (от 0 до 15 с)

$t(c)$	$S(m)$	$v (m/c)$	$t(c)$	$S(m)$	$v (m/c)$
0	0	0	8	200,1	35,6
1	4,8	9,6	9	235,9	36,0
2	18,7	17,9	10	272,1	36,3
3	40,1	24,4	11	308,5	36,4
4	66,9	28,9	12	345,0	36,5
5	97,4	31,9	13	381,5	36,6
6	130,3	33,8	14	418,1	36,6
7	164,7	35,0	15	454,7	36,6

Кроме таблицы необходимы графики зависимостей $v(t)$ и $S(t)$; по ним хорошо видно, как меняются со временем скорость и перемещение, т.е. приходит качественное понимание процесса.

Еще один элемент наглядности может внести изображение падающего тела через равные промежутки времени. Ясно, что при стабилизации скорости расстояния между изображениями станут равными. Можно прибегнуть и к цветовой раскраске - приему научной графики, описанному выше.

Наконец, можно запрограммировать звуковые сигналы, которые подаются через каждый фиксированный отрезок пути, пройденный телом - скажем, через каждый метр или каждые 100 метров - смотря по конкретным обстоятельствам. Надо выбрать интервал так, чтобы вначале сигналы были редкими, а потом, с ростом скорости, сигнал слышался все чаще, пока промежутки не сравняются. Таким образом, восприятию помогают элементы мультимедиа. Поле для фантазии здесь велико.

Приведем конкретный пример решения задачи о свободно падающем теле. Герой знаменитого фильма «Небесный тихоход» майор Булочкин, упав с высоты 6000 м в реку без парашюта, не только остался жив, но даже смог снова летать. Попробуем понять, возможно ли такое на самом деле или же подобное случается только в кино. Учитывая сказанное выше о математическом характере задачи, выберем путь численного моделирования. Итак, математическая модель выражается системой дифференциальных уравнений

$$\begin{cases} \frac{dh}{dt} = v, \\ \frac{dv}{dt} = \frac{mg - k_1 v - k_2 v^2}{m}. \end{cases} \quad (7.10)$$

Разумеется, это не только абстрактное выражение обсуждаемой физической ситуации, но и сильно идеализированное, т.е. ранжирование факторов перед построением математической модели произведено. Обсудим, нельзя ли произвести дополнительное ранжирование уже в рамках самой математической модели с учетом конкретно решаемой задачи, а именно - будет ли влиять на полет парашютиста линейная часть силы сопротивления и стоит ли ее учитывать при моделировании.

Так как постановка задачи должна быть конкретной, мы примем соглашение, каким образом падает человек. Он - опытный летчик и наверняка совершал раньше прыжки с парашютом, поэтому, стремясь уменьшить скорость, он падает не «солдатином», а лицом вниз, «лежа», раскинув руки в стороны. Рост человека возьмем средний - 1,7 м, а полубхват грудной клетки выберем в качестве характерного расстояния - это приблизительно 0,4 м. Для оценки порядка величины линейной составляющей силы сопротивления воспользуемся формулой Стокса. Для оценки квадра-

тичной составляющей силы сопротивления мы должны определиться со значениями коэффициента лобового сопротивления и площадью тела. Выберем в качестве коэффициента число $c = 1,2$ как среднее между коэффициентами для диска и для полусферы (выбор для качественной оценки правдоподобен). Оценим площадь: $S = 1,7 \cdot 0,4 = 0,7 \text{ (м}^2\text{)}$.

Выясним, при какой скорости сравниваются линейная и квадратичная составляющие силы сопротивления. Обозначим эту скорость v^{**} . Тогда

$$6\pi\mu r v^{**} = \frac{1}{2} c S \rho_{\text{среды}} v^{**2}$$

или

$$v^{**} = \frac{12\pi\mu r}{c S \rho_{\text{среды}}} = \frac{12 \cdot 3,14 \cdot 0,0182 \cdot 0,4}{1,22 \cdot 0,7 \cdot 1,29} \approx 0,2 \text{ м/с.}$$

Ясно, что практически с самого начала скорость падения майора Булочкина гораздо больше, и поэтому линейной составляющей силы сопротивления можно пренебречь, оставив лишь квадратичную составляющую.

После оценки всех параметров можно приступить к численному решению задачи. При этом следует воспользоваться любым из известных численных методов интегрирования систем обыкновенных дифференциальных уравнений: методом Эйлера, одним из методов группы Рунге - Кутты, одним из многочисленных неявных методов. Разумеется, у них разная устойчивость, эффективность и т.д. - эти сугубо математические проблемы здесь не обсуждаются. Программа, реализующая метод Рунге - Кутты четвертого порядка, может быть взята из примера, приведенного в следующем параграфе или из какого-нибудь стандартного пакета математических программ.

Отметим, что существует немало программ, моделирующих простые физические процессы типа рассматриваемого. У них реализован, в той или иной мере профессионально, диалоговый интерфейс, позволяющий вводить параметры, получать на экране таблицы, графики, движущиеся изображения. Однако в них, как правило, остаются скрытыми физические законы, определяющие процесс, ограничения модели, возможности ее усовершенствования. Такие программы полезны скорее как сугубо иллюстративные.

Вычисления производились до тех пор, пока «безпарашютист» не опустился на воду. Примерно через 15 с после начала полета скорость стала постоянной и оставалась такой до приземления (рис. 7.7). Отметим, что в рассматриваемой ситуации сопротивление воздуха радикально меняет характер движения; при отказе от его учета график скорости, изображенный на рисунке, заменился бы касательной к нему в начале координат.

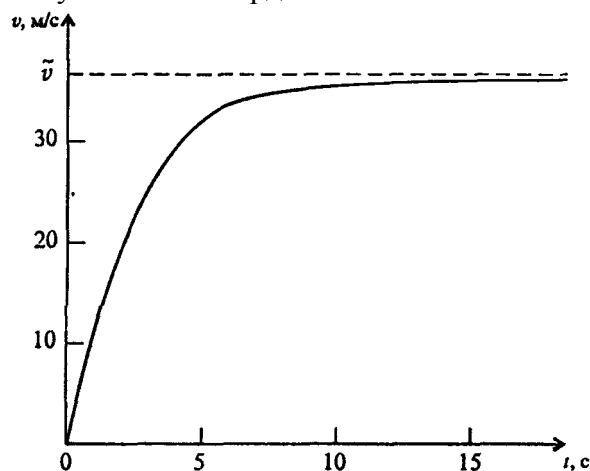


Рис. 7.7. График зависимости скорости падения «безпарашютиста» от времени

В некоторых случаях для ускорения процесса работы над какой-либо задачей целесообразно вместо составления программы воспользоваться готовой прикладной программой (например, табличным процессором). Покажем это на примере рассматриваемой задачи. В табл. 7.3 представлен небольшой фрагмент из табличного процессора Excel. Решение находится с помощью, так

называемого, исправленного метода Эйлера - одного из возможных вариантов метода Рунге - Кутты второго порядка.

Кроме того, в ячейках D2, D4, D6 в таблице будем хранить соответственно значения шага вычислений, массы «безпарашютиста», величины mg . Это связано с тем, что все константы также удобно хранить в отдельных ячейках, чтобы в случае их изменения не пришлось переписывать расчетные формулы. Достаточно записать

Таблица 7.3

Фрагмент таблицы, где представлено решение задачи о «безпарашютисте»

	A	B
1	t	v
2		
3	0	0
4	=СУММ(A3; D2)	=B3+D2/2* ((D6-D8*B3^2)/D4+(D6-D8*(B3+D2*(D6-D8*B3^2)/D4)^2)/D4)
5	=СУММ(A4; D2)	=B4+D2/2* ((D6-D8*B4^2)/D4+(D6-D8*(B4+D2*(D6-D8*B4^2)/D4)^2)/D4)
6	=СУММ(A5; D2)	=B5+D2/2* ((D6-D8*B5^2)/D4+(D6-D8*(B5+D2*(D6-D8*B5^2)/D4)^2)/D4)
7	=СУММ(A6; D2)	=B6+D2/2* ((D6-D8*B6^2)/D4+(D6-D8*(B6+D2*(D6-D8*B6^2)/D4)^2)/D4)
8	=СУММ(A7; D2)	=B7+D2/2* ((D6-D8*B7^2)/D4+(D6-D8*(B7+D2*(D6-D8*B7^2)/D4)^2)/D4)

формулу правильно один раз, а затем скопировать в остальные ячейки, при этом, как известно, она «настраивается» на соответствующую ячейку.

Таблица 7.4

Результаты вычислений, выполненных в табличном процессоре

	A	B	C	D
1	t	v		H
2				0,001
3	0	0		m
4	0,001	0,00981		80
5	0,002	0,01962		m*g
6	0,003	0,02943		784,8
7	0,004	0,03924		k2
8	0,005	0,04905		0,55083
9	0,006	0,05886		

Следует заметить, что для хранения результатов расчетов в данном случае требуется очень много ячеек таблицы, и хотя современные табличные процессоры позволяют хранить большой объем информации, в случае нехватки памяти рекомендуется увеличить шаг, с которым проводятся вычисления (при этом пожертвуем точностью вычислений). Табличный процессор позволяет представлять результаты расчетов и в графической форме. Можно при работе над задачей получить результаты двумя способами: с помощью табличного процессора и составлением собственной программы - для того, чтобы затем сравнить эти результаты и временные затраты каждого из способов. Но, несмотря на успешное применение табличного процессора при решении простейшей учебной задачи, следует признать, что для решения более громоздких в вычислительном плане задач предпочтительнее программировать самим. А теперь ответим на вопрос, поставленный в задаче. Известен такой факт: один из американских каскадеров совершил прыжок в воду с высоты 75 м (Бруклинский мост), и скорость приземления была 33 м/с. Сравнение этой величины с получившейся у нас конечной скоростью 37,76 м/с позволяет считать описанный в кинофильме эпизод вполне возможным. Обсуждаемой модели можно придать черты оптимизационной, поставив задачу так: парашютист прыгает с некоторой высоты и летит, не открывая парашюта; на какой высоте (или через какое время) ему следует открыть парашют, чтобы иметь к моменту приземле-

ния безопасную скорость? Или по-другому: как связана высота прыжка с площадью поперечного сечения парашюта (входящей в k_2), чтобы скорость приземления была безопасной? Выполнение таких исследований многократно более трудоемко, нежели просто изучение одного прыжка при заказанных условиях.

3.3. ДВИЖЕНИЕ ТЕЛА, БРОШЕННОГО ПОД УГЛОМ К ГОРИЗОНТУ. ЗАКОНЫ ПОДОБИЯ

Рассмотрим эту известную задачу с учетом сопротивления воздуха. Будучи брошенным под углом α к горизонту с начальной скоростью v_0 , тело летит, если не учитывать сопротивления воздуха, по параболе, и через некоторое время падает на землю. Напомним элементарное решение этой задачи. Разложим скорость на горизонтальную и вертикальную составляющие:

$$v_x^{(0)} = v_0 \cos \alpha, \quad v_y^{(0)} = v_0 \sin \alpha.$$

Поскольку движение по вертикали происходит под действием постоянной силы тяжести, то оно является равнозамедленным до достижения верхней точки на траектории и равноускоренным - после нее; движение же по горизонтали является равномерным. Из формул равноускоренного движения $v_y = v_y^{(0)} - gt$; раз в верхней точке $v_y = 0$, то время достижения верхней точки на траектории

$$\tilde{t} = \frac{v_y^{(0)}}{g} = \frac{v_0 \sin \alpha}{g}.$$

Высота этой точки

$$h = v_y^{(0)} \tilde{t} - \frac{g \tilde{t}^2}{2} = \frac{v_0^2 \sin^2 \alpha}{g} - \frac{g v_0^2 \sin^2 \alpha}{2g^2} = \frac{v_0^2 \sin^2 \alpha}{2g}.$$

Полное время движения до падения на землю $2\tilde{t}$; за это время, двигаясь равномерно вдоль оси x со скоростью $v_x^{(0)}$, тело пройдет путь

$$l = v_x^{(0)} \tilde{t} \cdot 2\tilde{t} = v_0 \cos \alpha \cdot 2 \frac{v_0 \sin \alpha}{g} = \frac{v_0^2 \sin 2\alpha}{g}.$$

Для нахождения траектории достаточно из текущих значений x и y исключить t :

$$x = v_x^{(0)} \cdot t, \quad y = v_y^{(0)} \cdot t - \frac{gt^2}{2};$$

следовательно,

$$y = v_y^{(0)} \cdot \frac{x}{v_x^{(0)}} - \frac{g}{2} \frac{x^2}{v_x^{(0)2}} = \operatorname{tg} \alpha \cdot x - \frac{g}{2v_0^2 \cos^2 \alpha} x^2. \quad (7.11)$$

Уравнение (7.11) - уравнение параболы.

Полученные формулы могут, в частности, послужить для тестирования будущей компьютерной программы. При достаточно большой начальной скорости сопротивление воздуха может значительно изменить характер движения. Прежде чем выписывать уравнения, вновь оценим, какая из составляющих силы сопротивления - линейная или квадратичная по скорости - дает больший вклад в эту силу, и нельзя ли одной из этих составляющих пренебречь. Оценку проведем для шарика; по порядку величины оценка не зависит от формы тела. Итак, шарик радиусом $r \approx 0,1$ м, движущийся со скоростью ~ 1 м/с, испытывает в воздухе линейную (стоксову) силу сопротивле-

$$F_1 = 6\pi\mu r v \approx 6\pi \cdot 0,0182 \cdot 0,1 \cdot 1 \text{ Н} \approx 3 \cdot 10^{-2} \text{ Н}$$

и квадратичную силу сопротивления

$$F_2 = \frac{cS\rho v^2}{2} \approx \frac{0,4\pi \cdot 0,1^2 \cdot 129 \cdot 1}{2} \text{ Н} \approx 8 \cdot 10^{-3} \text{ Н}.$$

Величины F_1 и F_2 сопоставимые (как принято говорить, «одного порядка», так как они различаются менее, чем в 5 раз). При увеличении размера тела F_2 растет быстрее, чем F_1 ($F_1 \sim r$, $F_2 \sim r^2$), при увеличении скорости F_2 также растет быстрее, чем F_1 ($F_1 \sim v$, $F_2 \sim v^2$). Таким образом, если мы моделируем движение брошенного мяча, камня, то необходимо в уравнениях удерживать обе составляющие силы сопротивления, но если мы захотим моделировать полет снаряда, выпущенного из орудия, где скорость полета почти на всем его протяжении сотни метров в секунду, то линейной составляющей силы сопротивления можно пренебречь. Проецируя уравнение $\frac{d\vec{v}}{dt} = \frac{\vec{F}}{m}$ на оси x и y , получаем

$$\frac{dv_x}{dt} = \frac{F_{\text{соп}}^x}{m}, \quad \frac{dv_y}{dt} = \frac{-mg + F_{\text{соп}}^y}{m}.$$

Поскольку в каждой точке траектории сила сопротивления направлена по касательной к траектории в сторону, противоположную движению, то

$$F_{\text{соп}}^x = -F_{\text{соп}} \cos \theta = -F_{\text{соп}} \frac{v_x}{v} = -(k_1 v + k_2 v^2) \frac{v_x}{v},$$

$$F_{\text{соп}}^y = -F_{\text{соп}} \sin \theta = -F_{\text{соп}} \frac{v_y}{v} = -(k_1 v + k_2 v^2) \frac{v_y}{v},$$

где θ - угол между текущим направлением скорости и осью x . Подставляя это в уравнение и учитывая, что $v = \sqrt{v_x^2 + v_y^2}$, получаем уравнения движения в переменных v_x , v_y .

$$\begin{cases} \frac{dv_x}{dt} = -\frac{k_1 + k_2 \sqrt{v_x^2 + v_y^2}}{m} v_x, \\ \frac{dv_y}{dt} = -g - \frac{k_1 + k_2 \sqrt{v_x^2 + v_y^2}}{m} v_y. \end{cases} \quad (7.12)$$

Поскольку представляет несомненный интерес и траектория движения, дополним систему (7.12) еще двумя уравнениями

$$\frac{dx}{dt} = v_x, \quad \frac{dy}{dt} = v_y, \quad (7.13)$$

и, решая их совместно с (7.12), будем получать разом четыре функции: $v_x(t)$, $v_y(t)$, $x(t)$, $y(t)$.

Прежде чем дать пример решения обсуждаемой задачи, покажем очень полезный прием, чрезвычайно популярный в физическом моделировании, называемый *обезразмериванием*. При решении конкретных задач мы пользуемся определенной системой единиц (СИ), в которой далеко не все числовые значения лежат в удобном диапазоне. Кроме того, абсолютные значения величин дают мало информации для качественного понимания. Скорость 15 м/с - много это или мало? Все дело в том, по сравнению с чем. Именно в сравнении с чем-то привычным и понятным мы обычно и воспринимаем слова «много» и «мало», даже если делаем это бессознательно. Идея обезразме-

ривания заключается в переходе от абсолютных значений расстояний, скоростей, времен и т.д. к относительным, причем отношения строятся к величинам, типичным для данной ситуации. В рассматриваемой задаче это особенно хорошо просматривается. В самом деле, при отсутствии сопротивления воздуха мы имеем значения l , h , t , определенные выше; сопротивление воздуха изменит характер движения, и если мы введем в качестве переменных величины

$$X = \frac{x}{l}, \quad Y = \frac{y}{h}, \quad \tau = \frac{t}{\tilde{t}}$$

- безразмерные расстояния по осям и время, - то при отсутствии сопротивления воздуха эти переменные будут изменяться в диапазоне от 0 до 1, а в задаче с учетом сопротивления отличия их максимальных значений от единицы ясно характеризуют влияние этого сопротивления. Для скоростей естественно ввести безразмерные переменные, соотнося проекции скорости на оси x и y с начальной скоростью v_0 :

$$V_x = \frac{v_x}{v_0}, \quad V_y = \frac{v_y}{v_0}.$$

Покажем, как перейти к безразмерным переменным в одном из наших уравнений, например, во втором уравнении системы (7.12). Имеем:

$$\frac{dv_x}{dt} = \frac{d(v_0 \cdot V_x)}{d(\tilde{t} \cdot \tau)} = \frac{v_0}{\tilde{t}} \cdot \frac{dV_x}{d\tau}$$

(так как постоянный множитель можно вынести за знак производной). Подставляя это в уравнение, получаем

$$\frac{v_0}{\tilde{t}} \cdot \frac{dV_y}{d\tau} = -g - \frac{k_1 + k_2 \sqrt{v_0^2 V_x^2 + v_0^2 V_y^2}}{m} v_0 V_y,$$

или

$$\frac{dV_y}{d\tau} = -\frac{\tilde{t}g}{v_0} - \frac{\tilde{t}}{v_0} \frac{k_1 + k_2 \sqrt{v_0^2 V_x^2 + v_0^2 V_y^2}}{m} v_0 V_y.$$

Подставляя

$$\tilde{t} = \frac{v_0 \sin \alpha}{g},$$

получаем

$$\frac{dV_y}{d\tau} = -\sin \alpha - a \cdot \sin \alpha \cdot V_y - b \cdot \sin \alpha \cdot \sqrt{V_x^2 + V_y^2} \cdot V_y,$$

где безразмерные комбинации параметров, входящих в исходные уравнения,

$$a = \frac{k_1 v_0}{mg}, \quad b = \frac{k_2 v_0^2}{mg}.$$

Выполним обезразмеривание во всех уравнениях (7.12), (7.13) (рекомендуем читателям проделать эту процедуру самостоятельно). В результате получим

$$\begin{cases} \frac{dV_x}{d\tau} = -a \cdot \sin \alpha \cdot V_x - b \sin \alpha \sqrt{V_x^2 + V_y^2} \cdot V_x, \\ \frac{dV_y}{d\tau} = -\sin \alpha - a \cdot \sin \alpha \cdot V_y - b \sin \alpha \sqrt{V_x^2 + V_y^2} \cdot V_y, \\ \frac{dX}{d\tau} = \frac{V_x}{2 \cos \alpha}, \\ \frac{dY}{d\tau} = \frac{2V_y}{\sin \alpha}. \end{cases} \quad (7.14)$$

Начальные условия для безразмерных переменных таковы:

$$V_x(0) = \cos \alpha, \quad V_y(0) = \sin \alpha, \quad X(0) = 0, \quad Y(0) = 0.$$

Важнейшая роль обезразмеривания - установление законов подобия. У изучаемого движения есть множество вариантов, определяемых наборами значений параметров, входящих в уравнения (7.12), (7.13) или являющихся для них начальными условиями: k_1, k_2, m, g, v_0, a . После обезразмеривания переменных появляются безразмерные комбинации параметров - в данном случае a, b, α - фактически определяющие характер движения. Если мы изучаем два разных движения с разными размерными параметрами, но такие, что a, b и α одинаковы, то движения будут качественно одинаковы. Число таких комбинаций обычно меньше числа размерных параметров (в данном случае вдвое), что также создает удобство при полном численном исследовании всевозможных ситуаций, связанных с этим процессом. Наконец, как уже отмечалось, величины V_x, V_y, X, Y, τ физически легче интерпретировать, чем их размерные аналоги, так как они измеряются относительно величин, смысл которых очевиден. Прежде чем предпринимать численное моделирование, отметим, что при учете лишь линейной составляющей силы сопротивления модель допускает аналитическое решение. Система уравнений (7.14) при $b = 0$ достаточно элементарно интегрируется и результаты таковы:

$$\begin{cases} V_x(\tau) = \cos \alpha \cdot e^{-a \sin \alpha \tau}, \\ V_y(\tau) = \sin \alpha \left(1 + \frac{1}{a \sin \alpha} \right) \cdot e^{-a \sin \alpha \tau} - \frac{1}{a}, \\ X(\tau) = \frac{1}{2a \sin \alpha} (1 - e^{-a \sin \alpha \tau}), \\ Y(\tau) = \frac{2}{a} \left(1 + \frac{1}{a \sin \alpha} \right) (1 - e^{-a \sin \alpha \tau}) - \frac{2}{a} \tau. \end{cases} \quad (7.14)$$

Исключая из двух последних формул время, получаем уравнение траектории:

$$Y(X) = 4 \left(1 + \frac{1}{a \sin \alpha} \right) X + \frac{2}{a^2 \sin^2 \alpha} \ln(1 - 2a \sin \alpha X).$$

Заметим, что эта формула не из тех, которые привычно визуализируются, например, по сравнению с совершенно отчетливой формулой (7.11), и здесь компьютер может быть полезен в том, чтобы составить ясное представление о влиянии линейной части силы сопротивления на изучаемое движение.

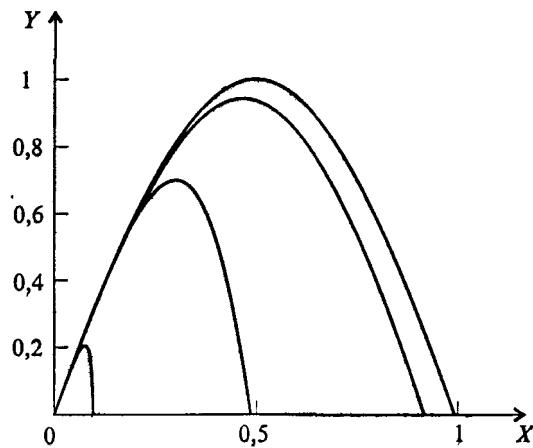


Рис. 7.8. Семейство траекторий при $\alpha = 45^\circ$ и значениях α , равных 0,01; 0,1; 1 и 10 (кривые - справа налево)

На рис. 7.8 приведены траектории четырех движений с разными значениями параметра α , характеризующего трение. Видно, как сильно оно влияет на движение - его форму, расстояния по вертикали и горизонтали. Общее исследование при произвольных значениях a и b поможет выполнить приведенная ниже программа.

Фактически представлены две программы: при активизации первого или второго блока. В первом случае она выдает результаты численного моделирования в виде таблицы значений безразмерных скоростей и координат при фиксированном наборе параметров a , b и α , значения которых устанавливаются в разделе определения констант. При взятии в фигурные скобки первого блока и активизации второго (т.е. снятия фигурных скобок) программа выдает в графическом режиме семейство траекторий, отличающихся значениями одного из трех безразмерных параметров (в данном случае b).

Программа 148. Реализация модели «Полет тела, брошенного под углом к горизонту»

```

Program Pod Uglom;
Uses Crt, Graph;
Type G = Array[1..4] Of Real;
Const A = 0; B = 0.1; (параметры модели)
      Al = Pi / 4; (угол - параметр модели)
      H = 0.001; Hpr = 0.1; (шаг интегрирования и шаг вывода результатов)
Var N, I, J, M, L, K : Integer;
Y0, Y : G; X0, X, Xpr, Al, Bl, Cosinus, Sinus : Real; LS : String;
Function Ff(I : Integer; X : Real; Y : G) : Real;
{описание правых частей дифференциальных уравнений}
Begin
  Case I Of
    1: Ff := -Al * Sinus * Y[1] - Bl * Sinus * Sqrt(Sqr(Y[1]) + Sqr(Y[2])) * Y[1];
    2: Ff := -Sinus - Al * Sinus * Y[1] - Bl * Sinus * Sqrt(Sqr(Y[1]) + Sqr(Y[2])) * Y[2];
    3: Ff := Y[1] / (2 * Cosinus);
    4: Ff := 2 * Y[2] / Sinus
  End
End;
End;
Procedure Runge_Kut (N: Integer; Var X: Real; Y0: G; Var Y: G; H: Real);
(метод Рунге-Кутты четвертого порядка)
Var I : Integer; Z, K1, K2, K3, K4 : G;
Procedure Right(X : Real; Y : G; Var F : G) ;
{вычисление правых частей дифференциальных уравнений}
Var I : Integer;
Begin
  For I := 1 To N Do F[I] := Ff(I, X, Y)
End;
Begin Right(X, Y0, K1); X := X + H / 2;

```

```

For I := 1 To N Do Z[I]:=Y0[I]+H*K1[I]/2; Right(X, Z, K2);
For I := 1 To N Do Z[I]:=Y0[I]+H*K2[I]/2; Right(X, Z, K3); X:=X+H/2;
For I := 1 To N Do Z[I] := Y0[I] + H * K3 [I]; Right (X, Z, K4);
For I := 1 To N Do
    Y[I]:=Y0[I]+H*(K1[I]+2*K2[I]+2*K3[I]+K4[I])/6;
End;
{следующий блок - для получения численных результатов при одном наборе
параметров}
{Begin
    Sinus := Sin(A1); Cosinus := Cos(A1); A1 := A; B1 := B; ClrScr;
    N:=4; X0:=0; Y0[1]:=Cosinus; Y0[2]:=Sinus; Y0[3]:=0; Y0[4]:=0;
    WriteLn(' время скорость координаты ');
    WriteLn; X := X0; Xpr := 0; Y[4] := Y0[4];
    While Y[4] >= 0 Do
        Begin
            If X >= Xpr Then
                Begin
                    WriteLn ('t=', X : 6 : 3, ' Vx=', Y0[1] : 6 : 3, ' Vy=',
                        Y0[2] : 6 : 3, ' X=', Y0[3] : 6 : 3, ' Y=', Y0[4] : 6 : 3) ;
                    Xpr := Xpr + Hpr
                End;
            Runge_Kut(N, X, Y0, Y, H); Y0 := Y
        End;
        WriteLn; WriteLn('для продолжения нажмите любую клавишу');
        Repeat Until KeyPressed
    End.}
{следующий блок - для изображения траекторий при не-
скольких наборах параметров}
Begin
    DetectGraph (J, M); InitGraph (J, M, '');
    L := 1; A1 := A; B1 := B; Sinus := Sin(A1); Cosinus := Cos(A1);
    While L < 5 Do
        Begin
            N := 4; (Количество уравнений в системе)
            X0 := 0; Y0[1] := Cosinus; (Начальные условия)
            Y0[2] := Sinus; Y0[3] := 0; Y0[4] := 0;
            SetColor(L); Line(400, 50 + 20 * (L - 1), 440, 50 + 20 * (L - 1));
            OutTextXY(450, 50 + 20 * (L - 1), '1 = ');
            Str(L, LS); OutTextXY(480, 50+20*(L-1), LS); X:=X0; Y[4]:=Y0[4];
            While Y[4] >= 0 Do
                Begin
                    Runge_Kut(N, X, Y0, Y, H); Y0 := Y;
                    PutPixel (Abs(Trunc(Y0[3]*500)), GetMaxY-Abs(Trunc(Y0[4]*500)), L) ;
                End;
                B1 := B1 * 10; L := L + 1
            End;
            OutTextXY(10, 50, 'для продолжения нажмите любую клавишу');
            Repeat Until KeyPressed; CloseGraph
        End.

```

Приведем пример. Рассмотрим полет чугунного ядра радиуса $R=0,07$ м, выпущенного с начальной скоростью $v_0 = 60$ м/с под углом $\alpha = 45^\circ$ к поверхности Земли. Определим, какое расстояние пролетит ядро, на какую максимальную высоту оно поднимется, а также проследим, как изменяется скорость полета со временем. Будем решать обезразмеренные уравнения, чтобы сократить число параметров. Вычислим значения параметров a и b , после чего решим систему дифференциальных уравнений. Учтем, что плотность чугуна $\rho_{\text{чуг}} = 7800$ кг/м³.

$$a = \frac{k_1 v_0}{mg} = \frac{6\pi\mu R v_0}{\frac{4}{3}\pi R^3 \rho_{\text{чуг}} g} = \frac{9\mu v_0}{2r^2 \rho_{\text{чуг}} g} \approx 0,013,$$

$$b = \frac{k_2 v_0^2}{mg} = \frac{\frac{1}{2} C \pi R^2 \rho_{\text{возд}} v_0^2}{\frac{4}{3}\pi R^3 \rho_{\text{чуг}} g} = \frac{3C\mu v_0^2}{8r \rho_{\text{чуг}} g} \approx 0,049.$$

Расчеты повторялись, сначала с шагом 0,1, затем - вдвое меньшим и т.д. (хорошо известный эмпирический метод контроля точности при пошаговом интегрировании дифференциальных уравнений), пока не был получен приемлемый шаг, при котором достигается точность 10^{-3} . Ясно, что расчеты надо проводить до тех пор, пока ядро не достигнет земли, т.е. пока Y не станет равным 0. Результаты моделирования - на рис. 7.9. В рассмотренном выше примере сопротивление среды оказывает незначительное влияние на движение тела. Проведем сравнение движения одного и того же тела без учета сопротивления среды и с его учетом, если среда достаточно вязкая (рис. 7.10).

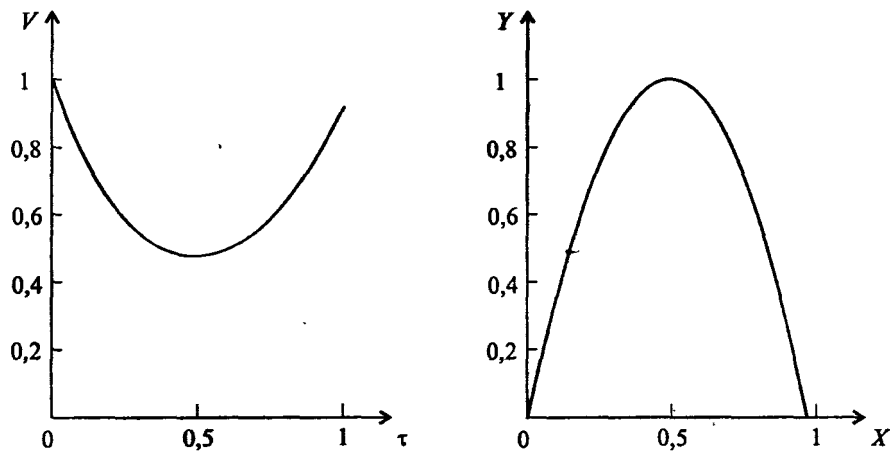


Рис. 7.9. Графики зависимости $V(\tau)$ и $Y(X)$ при решении задачи о полете ядра.

Безразмерное значение скорости V получается по формуле $V = \sqrt{V_x^2 + V_y^2}$.

Конечное значение скорости $V < 1$ вследствие сопротивления воздуха.

Траектория движения не является параболой по той же причине

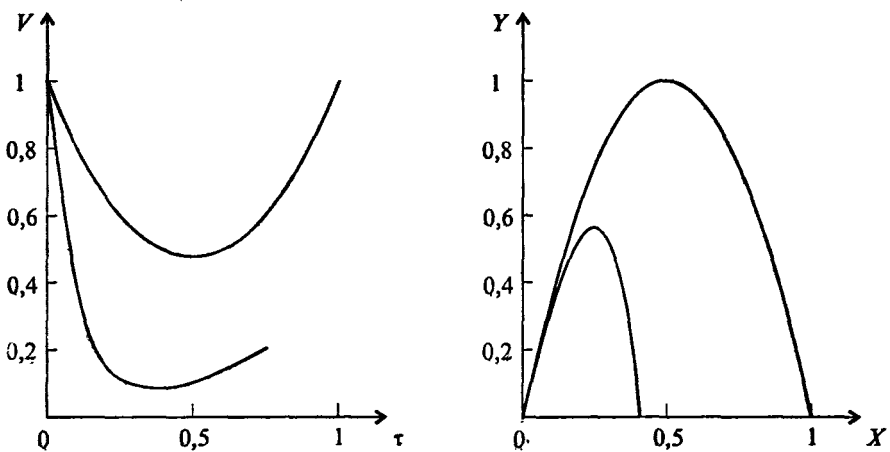


Рис. 7.10. Графики зависимости $V(\tau)$ и $Y(X)$ при решении задачи о полете тела, брошенного под углом к горизонту, без учета сопротивления воздуха (скорость изменяется от 1 и вновь достигает значения 1; траектория - парабола) и с учетом сопротивления воздуха (конечная скорость меньше 1, и траектория - далеко не парабола) ($a = 1$, $b = 1$)

С помощью приведенной выше программы можно провести полное исследование модели в широком диапазоне значений параметров и составить качественное представление об их влиянии на изучаемое движение. Некоторые результаты иллюстрируются рис. 7.11, 7.12.

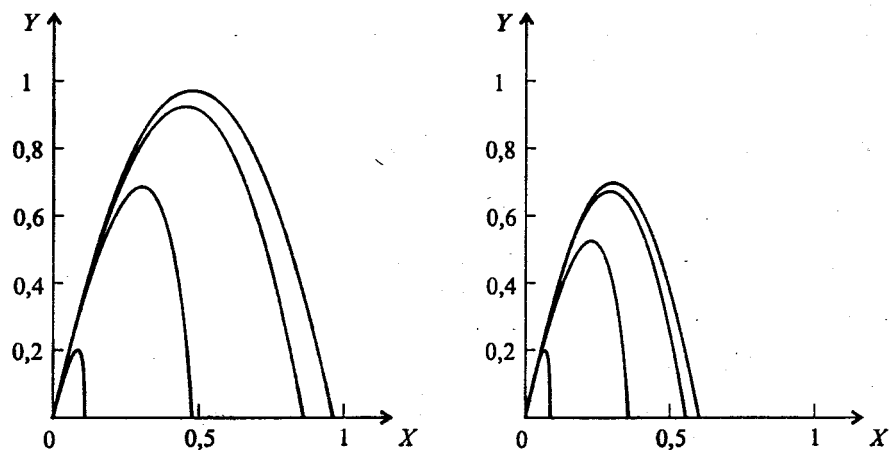


Рис. 7.11. Влияние параметра a на движение тела, брошенного под углом к горизонту, при $b = 0,1$ (слева) и при $b = 1$ (справа); $\alpha = \pi/4$ ($a = 0,01; 0,1; 1; 10$; кривые на рисунках соответственно располагаются справа налево)

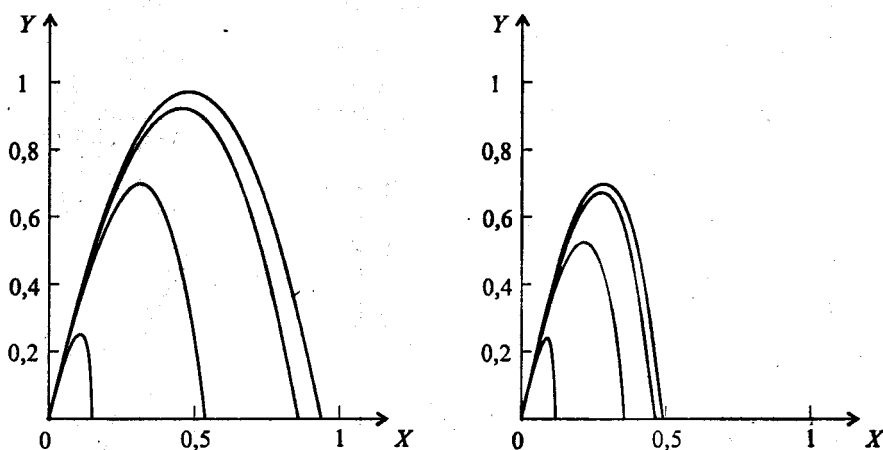


Рис. 7.12. Влияние параметра b на движение тела, брошенного под углом к горизонту, при $a = 0,1$ (слева) и при $a = 1$ (справа); $\alpha = \pi/4$ ($b = 0,01; 0,1; 1; 10$; кривые на рисунках соответственно располагаются справа налево)

3.4. ДВИЖЕНИЕ ТЕЛА С ПЕРЕМЕННОЙ МАССОЙ: ВЗЛЕТ РАКЕТЫ

Рассмотрим указанную задачу в максимально упрощенной постановке. Наши цели:

- а) достичь качественного понимания того, как скорость ракеты меняется во время взлета, как влияют на полет разные факторы;
- б) оценить оптимальное соотношение параметров, при котором ракета достигнет первой космической скорости и сможет вывести на орбиту полезный груз.

Таким образом, обсуждаемая модель имеет черты как дескриптивной, так и оптимизационной.

Взлет ракеты - сложный процесс, который неизбежно следует огрубить в попытке получения относительно простых и качественно верных результатов. Например, примем, что сила тяги двигателя - величина постоянная на всем этапе разгона. Реально это, скорее всего, не так. Но при упрощенном анализе колебаниями силы тяги пренебрежем, равно как и влиянием случайных порывов ветра и множеством других случайных и неслучайных факторов. Но при таком, даже самом упрощенном, анализе нельзя пренебречь наличием сопротивления воздуха, которое при высоких скоростях очень велико. Ни в коем случае нельзя пренебречь и убыванием массы ракеты в процессе взлета - оно огромно и составляет большую часть исходной массы. Так, у одной из крупнейших отечественных ракет «Энергия» стартовая масса составляет 20000 тонн, а к концу взлета всего 200 тонн.

Поиск математического описания проблем не составляет - в его основе все тот же второй закон Ньютона. Поскольку ракета очень быстро набирает столь высокую скорость, что линейной составляющей силы сопротивления заведомо можно пренебречь, то $F_{\text{сопр}} = k_2 v^2$. Примем, что топливо расходуется равномерно вплоть до его полного выгорания, т.е.

$$m(t) = \begin{cases} m_0 - \alpha t, & \text{если } m(t) > m_{\text{кон}}, \\ m_{\text{кон}}, & \text{если } m(t) = m_{\text{кон}}, \end{cases}$$

где m_0 - начальная масса ракеты, $m_{\text{кон}}$ - конечная (т.е. масса полезного груза, выводимого на орбиту), α - расход топлива; это допущение согласуется с допущением о постоянной силе тяги. Уравнение движения принимает вид в проекции на вертикальную ось

$$\frac{dv}{dt} = \frac{F_{\text{тяги}} - m(t)g - k_2 v^2}{m(t)}. \quad (7.17)$$

Казалось бы, можно задаться некоторыми значениями величин $F_{\text{тяги}}$, m_0 , α , k_2 и проводить моделирование, но это была бы чисто формальная деятельность, не учитывающая еще одного важнейшего обстоятельства. Поскольку ракета взлетает на огромную высоту (сотни километров), ясно, что сила сопротивления в менее плотных слоях атмосферы не может быть такой же, как вблизи поверхности Земли (при равных скоростях). Действительно, в коэффициент k_2 входит величина r - плотность окружающей среды, которая на «космических» высотах во много раз меньше, чем вблизи поверхности. Заглянем в справочник: на высоте 5,5 км плотность воздуха вдвое меньше, чем у поверхности, на высоте 11 км - вчетверо и т.д. Математически зависимость плотности атмосферы от высоты хорошо передается формулой

$$\rho = \rho_0 \exp(-\beta h),$$

где $b = 1,29 \cdot 10^{-4}$ (h измеряется в метрах, ρ_0 - плотность вблизи поверхности Земли). Поскольку величина h меняется в ходе полета, уравнение для изменения $h(t)$ следует добавить к уравнению (7.17) и записать следующую систему дифференциальных уравнений:

$$\begin{cases} \frac{dv}{dt} = \frac{F_{\text{тяги}} - m(t)g - \frac{1}{2} c \rho_0 \cdot \exp(-\beta h) \cdot S v^2(t)}{m(t)}, \\ \frac{dh}{dt} = v(t). \end{cases} \quad (7.18)$$

Наша модель становится все более реалистической. Ее совершенствование можно продолжить - например, учесть наличие у ракеты нескольких ступеней, каждая из которых имеет свой запас топлива и тягу двигателя - считая, что после уменьшения массы до некоторого значения сила тяги скачком изменяется; оставим это для самостоятельных размышлений. Перед решением уравнений удобно обезразмерить переменные. Естественной характерной скоростью в данной задаче является первая космическая скорость $v^* \approx 7,8$ км/с, при которой возможен вывод на орбиту полезного груза; характерное время - момент полной выработки горючего

$$t^* = \frac{(m_0 - m_{\text{кон}})}{\alpha},$$

где $m_{\text{кон}}$ - масса груза. Реально t^* - две-три минуты. За характерную высоту можно взять, например, h^* - ту, на которой плотность атмосферы уменьшается в 10 раз (примерно 17 км). Последняя величина может показаться несколько произвольной (впрочем, она таковой и является), но все равно удобнее измерять расстояния в данной задаче относительно величины, равной нескольким километрам, чем в метрах в системе СИ. Итак, введя безразмерные переменные

$$V = \frac{v}{v^*}, \quad \tau = \frac{t}{t^*}, \quad H = \frac{h}{h^*},$$

после несложных преобразований получим уравнения

$$\begin{cases} \frac{dV}{d\tau} = \frac{1}{f(\tau)} \cdot [a - b \cdot f(\tau) - p \cdot \exp(-2,3026H) \cdot V^2], \\ \frac{dH}{d\tau} = eV, \end{cases} \quad (7.19)$$

где $f(\tau)$ - известная функция:

$$f(\tau) = \begin{cases} 1 - (1 - k)\tau, & \text{если } \tau \leq 1, \\ k, & \text{если } \tau > 1, \end{cases}$$

а безразмерные параметры a, b, p, e, k выражаются через исходные так:

$$a = \frac{F_{\text{тяги}} t^*}{m_0 v^*}, \quad b = \frac{t^* g}{v^*}, \quad p = \frac{\frac{1}{2} c \rho_0 S v^* t^*}{m_0}, \quad e = \frac{v^* t^*}{h^*}, \quad k = \frac{m_{\text{кон}}}{m_0}.$$

То, что $f(\tau)$ определяется двумя формулами, связано с наличием двух этапов полета: до и после выработки топлива. Безразмерное время, разделяющее эти этапы - $\tau = 1$; если к этому моменту безразмерная скорость $V \geq 1$, то первая космическая скорость достигнута, в противном случае - нет. Параметр a управляет режимом полета; если при достижении величиной V значения, равного единице, топливо еще не все выработано (т.е. $\tau < 1$), можно с этого момента либо положить $a = 0$ («выключить двигатель»), либо продолжать разгон - в зависимости от постановки задачи. Рис. 7.13 иллюстрирует влияние изменения параметра a на динамику взлета ракеты в рамках принятых выше предположений при фиксированных значениях остальных параметров.

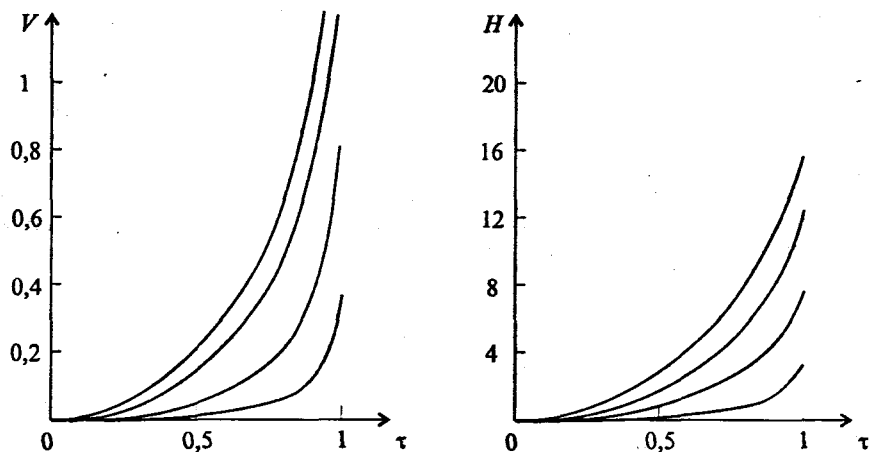


Рис. 7.13. Зависимости $V(\tau)$ и $H(\tau)$ при $a = 0,2, a = 0,3, a = 0,4$ и $a = 0,5$ (кривые на рисунках слева направо)

3.5. ДВИЖЕНИЕ НЕБЕСНЫХ ТЕЛ

Как движется Земля и другие планеты в пространстве? Что ждет комету, залетевшую из глубин космоса в Солнечную систему? Многовековая история поиска ответов на эти и другие вопросы о движении небесных тел хорошо известна; для многих людей, внесших большой вклад в науку, именно интерес к астрономии, устройству большого мира, был первым толчком к познанию.

По закону всемирного тяготения сила притяжения, действующая между двумя телами, пропорциональна их массам и обратно пропорциональна квадрату расстояния между ними. Если поместить начало системы координат на одном из тел (размерами тел по сравнению с расстоянием между ними будем пренебрегать), математическая запись силы, действующей на второе тело, имеет вид (рис. 7.14)

$$\vec{F} = -G \frac{Mm}{r^3} \vec{r}. \quad (7.20)$$

Здесь $G = 6,67 \cdot 10^{-11} \text{ м}^3/\text{кг} \cdot \text{с}^2$) - гравитационная постоянная.

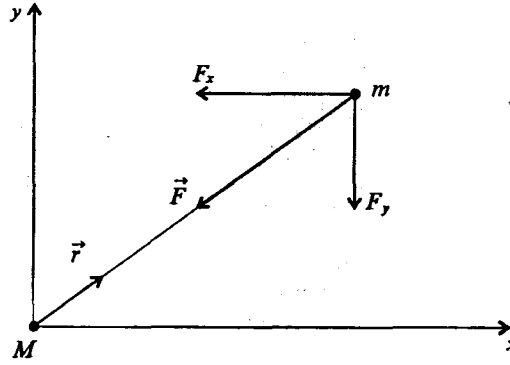


Рис. 7.14. Выбор системы координат при решении задачи двух тел

Знак «минус» в формуле (7.20) связан с тем, что гравитационная сила является силой притяжения, т.е. стремится уменьшить расстояние r между телами.

Далее мы ограничимся лишь изучением взаимного движения двух тел. При этом возникает непростой вопрос: с какой позиции (в какой системе координат) изучать это движение? Если делать это из произвольного положения - например, наблюдатель с Земли изучает взаимное движение Солнца и планеты Юпитер - задача станет для нас слишком сложной. Ограничимся лишь простейшей ситуацией: рассмотрим движение одного из тел с точки зрения наблюдателя, находящегося на втором, т.е., например, движения планеты или кометы относительно Солнца, Луны относительно Земли, пренебрегая при этом относительно небольшими силами притяжения от всех прочих небесных тел. Разумеется, мы тем самым произвели ранжирование факторов, и наши последующие действия имеют отношение к реальности лишь в меру соблюдения определенных условий.

Уравнение, описывающее движение тела m в указанной системе координат, имеет вид

$$m \frac{d^2 \vec{r}}{dt^2} = -G \frac{Mm}{r^3} \vec{r},$$

или в проекциях на оси x, y

$$\frac{d^2 x}{dt^2} = -GM \frac{x}{\sqrt{(x^2 + y^2)^3}}, \quad \frac{d^2 y}{dt^2} = -GM \frac{y}{\sqrt{(x^2 + y^2)^3}}. \quad (7.21)$$

Интересующая нас орбита сильно зависит от «начальной скорости» тела m и «начального расстояния». Мы взяли эти слова в кавычки, так как при изучении движения космических тел нет столь отчетливо выделенного «начального момента», как в ранее рассмотренных ситуациях. При моделировании нам придется принять некоторое положение условно за начало, а затем изучать движение дальше. Очень часто космические тела движутся практически с постоянной скоростью по орбитам, близким к круговым. Для таких орбит легко найти элементарное соотношение между скоростью и радиусом. В этом случае сила тяготения выступает в роли центростремительной, а центростремительная сила при постоянной скорости выражается известной из начального курса физики формулой mv^2/r . Таким образом, имеем

$$\frac{mv^2}{r} = G \frac{Mm}{r^2},$$

или

$$v = \sqrt{\frac{MG}{r}}, \quad (7.22)$$

- искомое соотношение.

Период движения по такой орбите

$$T = \frac{2\pi r}{v} = 2\pi \sqrt{\frac{r^3}{MG}}.$$

Заметим, что отсюда вытекает один из законов Кеплера, приведший Ньютона к открытию закона всемирного тяготения: отношение кубов радиусов орбит любых двух планет Солнечной системы равно отношению квадратов периодов их обращения вокруг Солнца, т.е. $\left(\frac{r_1}{r_2}\right)^3 = \left(\frac{T_1}{T_2}\right)^2$.

Более точная формулировка дана ниже (так как реально орбиты планет не вполне круговые). Если соотношение (7.22) нарушено, то орбита не будет круговой. Выяснить, какой она будет, можно в ходе численного моделирования. Сведем (7.21) к системе четырех дифференциальных уравнений первого порядка:

$$\begin{cases} \frac{dx}{dt} = v_x, \\ \frac{dv_x}{dt} = -GM \frac{x}{\sqrt{(x^2 + y^2)^3}}, \\ \frac{dy}{dt} = v_y, \\ \frac{dv_y}{dt} = -GM \frac{y}{\sqrt{(x^2 + y^2)^3}}. \end{cases} \quad (7.23)$$

В этой задаче особенно неудобно работать с размерными величинами, измеряемыми миллиардами километров, секунд и т.д. В качестве величин для обезразмеривания удобно принять характерное расстояние от Земли до Солнца $\rho = 1,496 \cdot 10^{11}$ м, (так называемая, астрономическая единица), период круговой орбиты $T = 2\pi \sqrt{\frac{\rho^3}{MG}}$, соответствующий этому расстоянию, скорость дви-

жения по ней $\bar{v} = \sqrt{\frac{MG}{\rho}}$, т.е. принять

$$X = \frac{x}{\rho}, \quad Y = \frac{y}{\rho}, \quad V_x = \frac{v_x}{\bar{v}}, \quad V_y = \frac{v_y}{\bar{v}}, \quad \tau = \frac{t}{T}.$$

После обезразмеривания получаем

$$\begin{cases} \frac{dX}{d\tau} = 2\pi V_x, \\ \frac{dV_x}{d\tau} = -2\pi \frac{X}{\sqrt{(X^2 + Y^2)^3}}, \\ \frac{dY}{d\tau} = 2\pi V_y, \\ \frac{dV_y}{d\tau} = -2\pi \frac{Y}{\sqrt{(X^2 + Y^2)^3}}. \end{cases} \quad (7.24)$$

Отметим замечательное обстоятельство: в безразмерных переменных уравнения вообще не содержат параметров! Единственное, что отличает разные режимы движения друг от друга - начальные условия.

Можно доказать, что возможные траектории движения, описываемые уравнениями (7.24) - эллипс, парабола и гипербола.

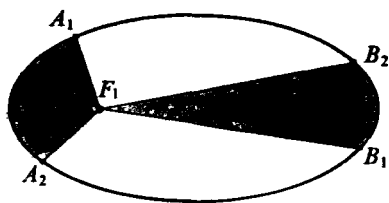


Рис. 7.15. Иллюстрация второго закона Кеплера

Напомним законы Кеплера, рис. 7.15.

1. Всякая планета движется по эллиптической орбите с общим фокусом, в котором находится Солнце.

2. Каждая планета движется так, что ее радиус-вектор за одинаковые промежутки времени описывает равные площади; на рисунке промежутки времени движения от A_1 к A_2 и от B_1 к B_2 считаются одинаковыми, а площади секторов $F_1A_1A_2$ и $F_1B_1B_2$ равны. Это означает, что чем ближе планета к Солнцу, тем у нее больше скорость движения по орбите.

3. Отношение кубов больших полуосей орбит двух любых планет Солнечной системы равно отношению квадратов периодов их обращения вокруг Солнца.

Уравнения (7.24) описывают движение не только планет, но и любых тел, попадающих в поле тяготения большой массы. Так, в Солнечной системе существует огромное количество комет, движущихся по чрезвычайно вытянутым эллиптическим орбитам с периодами от нескольких земных лет до нескольких миллионов земных лет. Судьбы небесных тел, не являющихся постоянными членами Солнечной системы, а залетевших в нее издалека, определяются их скоростью - если она достаточно велика, то орбита будет гиперболической, и, облетев Солнце, тело покинет Солнечную систему, если нет - перейдет на эллиптическую орбиту и станет частью системы; пограничная между ними орбита - параболическая.

Все эти утверждения можно проверить и детально исследовать с помощью уравнений (7.24). При этом полезно и удобно использовать одно важнейшее свойство обсуждаемой системы, которого не было у рассмотренных ранее - сохранение полной энергии движущегося тела (такое свойство называется «консервативность»). Полная энергия движущегося небесного тела m в системе двух тел имеет значение

$$E = \frac{1}{2}mv^2 - G\frac{mM}{r}.$$

Первое слагаемое - кинетическая, второе - потенциальная энергия. В безразмерных переменных

$$\varepsilon = \frac{E}{\left(\frac{mMG}{\rho}\right)} = \frac{1}{2}\sqrt{(V_x^2 + V_y^2)} - \frac{1}{\sqrt{X^2 + Y^2}}.$$

Наличие неизменного параметра ε в ситуации, когда изменяются V_x , V_y , X , Y , позволяет контролировать процесс решения системы дифференциальных уравнений, проверять устойчивость метода, подбирать шаг интегрирования.

3.6. ДВИЖЕНИЕ ЗАРЯЖЕННЫХ ЧАСТИЦ

Закон Кулона, описывающий взаимодействие точечных зарядов, так похож на закон всемирного тяготения, что очевидна близость подходов к моделированию движения заряженной частицы в электростатическом поле и движения малого небесного тела в поле тяжести.

Напомним закон Кулона: между двумя зарядами Q и q разных (одинаковых) знаков действует сила притяжения (отталкивания)

$$\vec{F} = -\frac{1}{4\pi\epsilon_0} \cdot \frac{Qq}{r^3} \vec{r}, \quad (7.25)$$

где $\epsilon_0 = 8,85 \cdot 10^{-12}$ Ф/м - так называемая электрическая постоянная, подробности - в любом курсе физики; Ф/м - «фарада на метр»

Выбор в (7.25) знака «минус» соответствует тому же выбору координат и направлений, что на рис. 7.14, иллюстрирующем закон тяготения.

Первая из задач, которую можно рассмотреть - движение «малого» заряда некоторого знака в поле, создаваемом «большим» неподвижным зарядом другого знака. Эта задача после обезразмеривания уравнений (оставим его читателю) в точности та же, что и рассмотренная выше задача движения «малого» небесного тела. В электростатике, однако, есть возможность рассмотрения широкого круга задач, не имеющих аналога в гравитации. Перечислим простейшие из них:

- 1) движение «малого» заряда в поле «большого» при взаимном отталкивании;
- 2) движение заряженного тела в поле, созданном несколькими фиксированными зарядами произвольных знаков (рекомендуем начать со случая, когда все фиксированные заряды лежат в одной плоскости и начальное положение и скорость движущегося заряда - в той же плоскости);
- 3) движение заряженного тела между пластинами конденсатора (рекомендуем ограничиться плоским движением).

В последнем случае закон Кулона «в лоб» применить трудно - ведь заряженная пластина не может рассматриваться как «точечный заряд». При моделировании можно воспользоваться таким приемом: разбить пластину на несколько маленьких квадратиков, каждому из них приписать приходящийся на его долю заряд и заменить пластину эффективным набором «точечных» зарядов, взаимодействующих с пролетающей частицей. Этот прием - замена непрерывного дробным (дискретизация) обсуждается в следующих разделах.

Моделируя движение заряда, можно получать самые замысловатые траектории, помогающие, с одной стороны, лучше понять закон Кулона, а с другой - научиться визуализации динамических процессов на экране компьютера.

Для решения первой задачи рассмотрим сначала модель, характеризующую движение «малого» заряда в поле «большого», если заряды имеют разные знаки.

Получаем

$$\begin{cases} \frac{dx}{dt} = v_x, & \frac{dy}{dt} = v_y, \\ \frac{dv_x}{dt} = -\frac{1}{4\pi\epsilon_0} \cdot \frac{Qq}{m} \cdot \frac{x}{\sqrt{(x^2 + y^2)^3}}, & \frac{dv_y}{dt} = -\frac{1}{4\pi\epsilon_0} \cdot \frac{Qq}{m} \cdot \frac{y}{\sqrt{(x^2 + y^2)^3}}. \end{cases} \quad (7.26)$$

Как обычно, удобно провести обезразмеривание полученной системы. В качестве параметров, с помощью которых проводим обезразмеривание, можно выбрать те, которые характерны для движения «малого» заряда по круговой орбите. Предлагаем читателю самостоятельно проделать эту работу, после чего получаем систему дифференциальных уравнений, практически полностью совпадающую с (7.24), поэтому вновь выписывать здесь ее не будем.

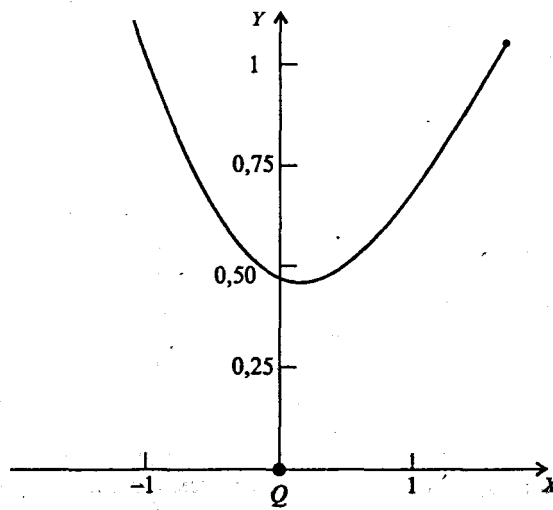


Рис. 7.16. Траектория движения малого положительного заряда в поле большого положительного заряда при $V_x(0) = -2$; $V_y(0) = -1$; $X(0) = 1,5$; $Y(0) = 1$

Возвращаясь к задаче, когда заряды являются одинаково заряженными и потому отталкиваются, можно заметить, что уравнения будут аналогичными, лишь во втором и в четвертом уравнениях знаки «минус» сменятся на «плюс».

В качестве примера на рис. 7.16 приведена типичная траектория движения при взаимном отталкивании зарядов.

3.7. КОЛЕБАНИЯ МАТЕМАТИЧЕСКОГО МАЯТНИКА

Колебательное движение - одно из самых распространенных в природе. Разнообразные маятники в часах и других технических устройствах, колебания мембран и оболочек, колебания атомов в молекулах, ионов и молекул в кристаллах и многие другие процессы в живой и неживой природе в чем-то схожи: объект движется таким образом, что многократно проходит через одни и те же точки, периодически воспроизводя одно и то же состояние. Изучив его движение на сравнительно коротком отрезке времени, включающем один период, мы можем составить полное представление о его движении в будущем (если оно не будет изменено вмешательством извне).

Хотя колебательные движения бывают весьма многообразны, их сущность можно постичь на нескольких относительно простых примерах. Остановимся на одном из самых простых, название которого вынесено в заголовок. Этот пример рассматривается в любом школьном курсе физики, но, располагая более совершенным математическим аппаратом и прибегая к компьютерному моделированию, можно продвинуться в изучении колебаний математического маятника дальше и понять закономерности колебательного движения глубже.

Рассмотрим идеализированную систему, состоящую из тела массы m , прикрепленного к нижнему концу жесткого «невесомого» стержня длиной l , верхний конец которого вращается без трения в точке подвеса, рис. 7.17.

Если груз отклонить от положения равновесия на угол θ_0 и отпустить, то «математический маятник» будет колебаться в вертикальной плоскости.

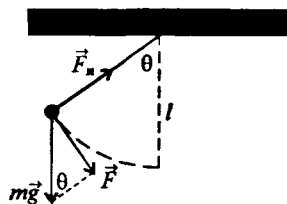


Рис. 7.17. Колебания математического маятника

Поскольку движение груза происходит по дуге окружности радиуса l , то его положение характеризуется в каждое мгновение углом θ . Линейная скорость и ускорение равны

$$v = l \frac{d\theta}{dt}, \quad a = l \frac{d^2\theta}{dt^2}. \quad (7.27)$$

На груз действуют две силы: сила тяжести $m\vec{g}$ и упругая сила натяжения стержня $\vec{F}_n$. При выводе уравнения движения достаточно учесть лишь компоненту силы $m\vec{g}$, направленную по касательной к дуге: $F = mg \sin \theta$, направлена она в сторону уменьшения θ . Сила $\vec{F}_n$ перпендикулярна к касательной и вклад в это уравнение не дает. Уравнение движения примет вид

$$ml \frac{d^2\theta}{dt^2} = -mg \sin \theta, \quad \text{или} \quad \frac{d^2\theta}{dt^2} = -\frac{g}{l} \sin \theta. \quad (7.28)$$

Обычно в курсе физики ограничиваются исследованием малых колебаний. Если $|\theta| \ll 1$, то уравнение (7.28) можно считать эквивалентным (так как $\sin \theta \approx \theta$; здесь и далее используется радианная мера углов) уравнению

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l} \theta.$$

Решение его элементарно:

$$\theta = A \cos \omega t + B \sin \omega t,$$

где $\omega = \sqrt{\frac{g}{l}}$ - собственная частота, $T = \frac{2\pi}{\omega}$ - период колебания маятника. Значения A и B зависят от начальных условий. Если при $t = 0$

$$\theta = \theta_0, \quad v = v_0,$$

то

$$\theta = \theta_0 \cos \omega t + \frac{v_0}{l\omega} \sin \omega t$$

или, как часто записывают,

$$\theta = A \cos(\omega t + \varphi),$$

где φ - так называемая, начальная фаза; A - амплитуда колебания; A и φ легко выразить через начальные условия θ_0 и v_0 .

$$A = \sqrt{\theta_0^2 + \frac{v_0^2}{(l\omega)^2}}, \quad \text{tg} = -\frac{v_0}{l\omega\theta_0}.$$

Движение, происходящее по закону (7.29), называют гармоническим колебательным движением. Слово «гармонический» связывают с простой тригонометрической функцией (синусом или косинусом); так, гармоническим является и движение $A \sin(\omega t + \varphi)$, к которому также можно свести (7.29) (оно отличается лишь сдвигом фазы на $\pi/2$).

Для изучения колебаний с большой амплитудой следует обратиться к уравнению (7.28), которое заведомо не интегрируется в элементарных функциях. Обезразмерим его, взяв за характерный масштаб времени период малого колебания. Если $\tau = t/T$, то

$$\frac{d^2\theta}{dt^2} = T^2 \left(-\frac{g}{l} \sin \theta \right), \text{ или } \frac{d^2\theta}{d\tau^2} = -4\pi^2 \sin \theta. \quad (7.30)$$

Это уравнение вообще не содержит параметров! Достаточно его решить, и мы составим полное представление о природе «больших» колебаний. В этом проявляется сила приема обезразмеривания.

Сведем (7.30) к системе двух уравнений первого порядка:

$$\begin{cases} \frac{d\theta}{d\tau} = x, \\ \frac{dx}{d\tau} = -4\pi^2 \sin \theta. \end{cases} \quad (7.31)$$

Существенно, что система консервативна, и полная энергия сохраняется (до тех пор, пока мы не учитываем трение и воздействие извне):

$$E = E_{\text{кин}} + E_{\text{пот}} = \frac{1}{2} ml^2 \left(\frac{d\theta}{dt} \right)^2 + mgl(1 - \cos \theta). \quad (7.32)$$

В безразмерных переменных x и θ

$$\varepsilon \equiv \frac{E}{mgl} = \frac{x^2}{8\pi^2} + 1 - \cos \theta. \quad (7.33)$$

Как и при моделировании движения небесных тел, сохранение ε в ходе интегрирования - прекрасный критерий для изучения устойчивости метода, выбора шага и т.д. На рис. 7.18 представлен график зависимости $\theta(\tau)$ для $\theta_0 = \pi/2$ и $v_0 = 0$ (сплошная линия). На первый взгляд, это косинусоида (7.29), но, во-первых, это не так (зрительным впечатлениям в таких случаях доверять особо не следует), а, во-вторых, у этого движения период отнюдь не определяется формулой, следующей из решения задачи о малых колебаниях. Для сравнения на рисунке представлено пунктирной линией гармоническое движение с той же амплитудой $\pi/2$, следующее из формального решения задачи о малых колебаниях (его период равен единице вследствие обезразмеривания).

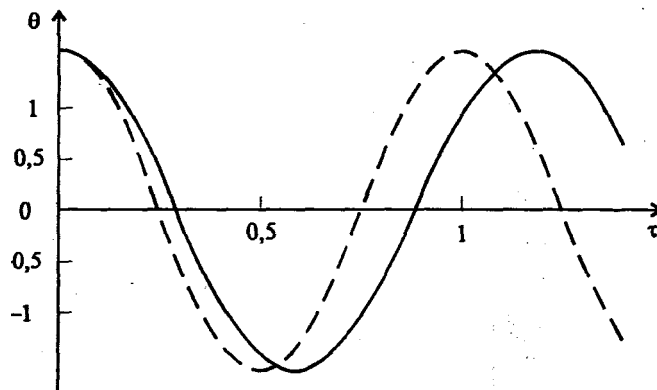


Рис. 7.18. Графики зависимости $\theta(\tau)$ для $\theta_0 = \pi/2$ и $v_0 = 0$ (сплошная линия) и гармонического движения с той же амплитудой $\pi/2$ (пунктирная линия)

Итак, реальный период, оказывается, зависит от амплитуды колебания вопреки тому, что предсказывает теория, основанная на приближении малых колебаний. Определить зависимость периода от амплитуды - относительно несложная задача для самостоятельного решения.

Вернемся снова к разговору о периодическом, но не гармоническом движении. Период колебаний в рассмотренном примере приблизительно равен 1,18 (определено в численном эксперименте). Уравнение гармонического движения с периодом T и амплитудой A

$$\theta = A \cos\left(\frac{2\pi}{T}t + \varphi\right),$$

(в нашем конкретном случае $A = \pi/2$, $T \approx 1,18$, $\varphi = 0$). В табл. 7.5 сведены результаты численного решения уравнений (7.31) (вторая строка) и табулирования функции при $A = \pi/2$, $T \approx 1,18$, $\varphi = 0$ (третья строка) на промежутке времени, чуть большем периода. Хотя различия и невелики, но видно, что движение не является гармоническим.

Таблица 7.5
Сравнение результатов моделирования с гармоническими колебаниями

t	0,0	0,1	0,2	0,3	0,4	0,5	0,6
$\theta_{\text{реал}}$	1,5708	1,3737	0,7971	-0,0437	-0,8688	-1,4104	-1,5689
$\theta_{\text{гарм}}$	1,5708	1,3533	0,7611	-0,0418	-0,8332	-1,3938	-1,5686
$\varphi(t)$	1,5710	1,3737	0,7938	-0,0473	-0,8696	-1,4077	-1,5631

t	0,7	0,8	0,9	1,0	1,1	1,2	1,3
$\theta_{\text{реал}}$	-1,3331	-0,7228	0,1308	0,9374	1,4434	1,5632	1,2889
$\theta_{\text{гарм}}$	-1,3090	-0,6870	0,1253	0,9028	1,4304	1,5619	1,2609
$\varphi(t)$	-1,3299	-0,7216	0,1297	0,9371	1,4448	1,5631	1,2869

Широчайшее распространение в математике и ее приложениях, связанных с периодическими функциями, имеет, так называемый, гармонический анализ. Для тех, кто не изучал соответствующий раздел математики, дадим представление о нем на данном примере. Поскольку тригонометрические функции, соответствующие гармоническому движению, хорошо изучены и привычны, то стремление передать периодическое (но не гармоническое) движение хотя бы суммой нескольких гармонических вполне понятно. Все эти «гармоники» должны иметь, естественно, тот же период, что и -изучаемая функция. Если ее период T , то, кроме тригонометрических функций $\sin \frac{2\pi}{T}t$, $\cos \frac{2\pi}{T}t$ период T имеют и функции с частотами, кратными $\omega = \frac{2\pi}{T}$, т.е. $\sin\left(k \frac{2\pi}{T}t\right)$, $\cos\left(k \frac{2\pi}{T}t\right)$ при любом целом $k > 0$. Гармоническое разложение функции $f(t)$ с периодом T в общем случае имеет вид

$$f(t) = a_0 + a_1 \cos\left(\frac{2\pi}{T}t\right) + b_1 \sin\left(\frac{2\pi}{T}t\right) + a_2 \cos\left(2\frac{2\pi}{T}t\right) + \\ + b_2 \sin\left(2\frac{2\pi}{T}t\right) + \dots + a_k \cos\left(k\frac{2\pi}{T}t\right) + b_k \sin\left(k\frac{2\pi}{T}t\right) + \dots,$$

причем число гармоник-слагаемых формально бесконечно велико. Те, кто изучал ряды Фурье, знают общие правила вычисления коэффициентов a_0 , a_1 , b_1 , a_2 , b_2 , ... Если ограничиться лишь небольшим числом гармоник, скажем, тремя, то коэффициенты можно приближенно найти интерполяцией. Взяв за узлы точки $t = 0$; 0,4 и 1 (выбор достаточно произволен) и решив систему трех линейных алгебраических уравнений, получим

$$f(t) \approx \varphi(t) = 1,592 \cos\left(\frac{2\pi}{T}t\right) + 0,003 \cos\left(\frac{4\pi}{T}t\right) - 0,024 \cos\left(\frac{6\pi}{T}t\right).$$

Значения функции $\varphi(t)$ приведены в четвертой колонке табл. 7.5; они значительно ближе к $\theta_{\text{реал}}$, чем $\theta_{\text{гарм}}$. Обратим внимание на то, что первый коэффициент значительно больше остальных, что еще раз подчеркивает, что движение близко к гармоническому.

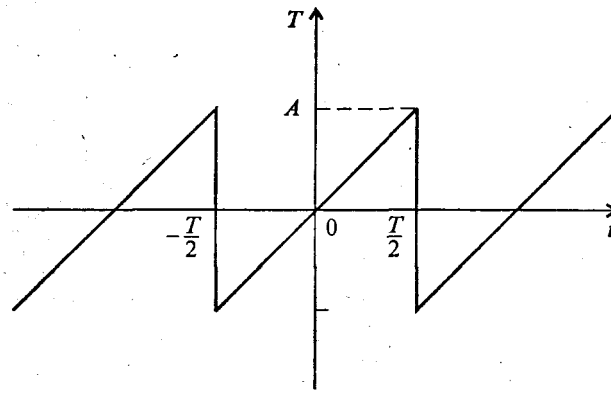


Рис. 7.19. Периодический сигнал, подобный вырабатываемому генератором строчной развертки в телевизоре

Отвлечемся ненадолго от данной конкретной задачи и еще раз подчеркнем, что гармоническому разложению доступна любая периодическая функция. Например, периодический сигнал пилообразной формы (рис. 7.19), похожий на тот, который вырабатывает генератор строчной развертки в телевизоре, имеет следующее спектральное разложение;

$$y = 2A \left(\sin \frac{2\pi}{T} t - \frac{1}{2} \sin \frac{4\pi}{T} t + \frac{1}{3} \sin \frac{6\pi}{T} t - \dots \right).$$

Здесь уже нет столь быстрого спада коэффициентов при гармониках, так как исходная линия вовсе не похожа на синусоиду. Попробуйте протабулировать сумму вначале первых двух гармоник, затем трех, четырех и т.д. и пронаблюдать, как по мере роста числа слагаемых сумма все больше похожа на исходную функцию.

Колебания маятника при наличии трения. Поскольку сила трения при малых скоростях пропорциональна скорости, а скорость $v = l \frac{d\theta}{dt}$, то уравнение свободных колебаний маятника с учетом трения выглядит так:

$$ml \frac{d^2\theta}{dt^2} = -mg \sin \theta - kl \frac{d\theta}{dt}.$$

Преобразуем его к виду

$$\frac{d^2\theta}{dt^2} + 2\kappa \frac{d\theta}{dt} + \omega^2 \sin \theta = 0, \quad (7.34)$$

где, как и выше, $\omega = \sqrt{\frac{g}{l}}$, а $k = \frac{k}{2m}$ (коэффициент 2 записан по традиции для $K / 2m$ удобства).

При малых колебаниях уравнение (7.34) превращается в

$$\frac{d^2\theta}{dt^2} + 2\kappa \frac{d\theta}{dt} + \omega^2 \theta = 0. \quad (7.35)$$

Его решение таково: затухающие колебания при $\kappa < \omega$ и затухание без колебаний при $\kappa \geq \omega$. Все это можно проверить в ходе численного моделирования, хотя уравнение (7.35) допускает аналитическое решение. Приведем его: при $\kappa < \omega$

$$\theta = A e^{-\kappa t} \cos(\omega_1 t + \varphi),$$

где $\omega_1 = \sqrt{\omega^2 - \kappa^2}$, A - амплитуда, φ - начальная фаза (A и φ легко выразить через начальные зна-

чения θ_0 и v_0). При $k \geq \omega$

$$\theta = Ae^{-(\kappa+\omega_1)t} + Be^{-(\kappa-\omega_1)t}$$

где A и B также можно выразить через θ_0 и v_0 .

Что же касается уравнения (7.34), то его аналитическое решение отсутствует, и при численном моделировании можно поставить ряд задач о том, насколько решения уравнений (7.34) и (7.35) различаются в зависимости от начальной амплитуды.

Вынужденные колебания. Если на маятник воздействует внешняя сила $F(t)$, меняющаяся со временем, то уравнения движения получаются из (7.34) добавлением $F(t)$ к правой части. Рассмотрим лишь случай периодического внешнего воздействия: $F(t) = F_0 \cos \lambda t$, где λ - частота вынуждающей силы. Имеем уравнение движения маятника:

$$\frac{d^2\theta}{dt^2} + 2\kappa \frac{d\theta}{dt} + \omega^2 \sin \theta = f \cos \lambda t, \quad (7.36)$$

где $f = \frac{F_0}{ml}$. При малой амплитуде результирующего движения уравнение (7.36) примет вид

$$\frac{d^2\theta}{dt^2} + 2\kappa \frac{d\theta}{dt} + \omega^2 \theta = f \cos \lambda t. \quad (7.37)$$

Движение, описываемое уравнением (7.37), состоит из двух этапов. На первом оно складывается из двух колебательных движений: затухающих собственных колебаний с частотой $\omega_1 = \sqrt{\omega^2 - \kappa^2}$ (при $\kappa < \omega$) и вынужденных колебаний с частотой λ . На втором этапе, по истечении времени $t \gg 1/\kappa$, остаются лишь вынужденные периодические колебания, амплитуда которых зависит от соотношения частот λ и ω_1 и резко возрастает при $\lambda \approx \omega_1$ - явление резонанса, описанное в любом учебнике физики. Численное интегрирование уравнения (7.37) необязательно, так как решение можно записать в виде формул, содержащих лишь элементарные функции:

$$\begin{aligned} \theta(t) = & Ae^{-\kappa t} \sin \omega_1 t + Be^{-\kappa t} \cos \omega_1 t + \\ & + \frac{f(\lambda^2 - \omega^2)}{4\kappa^2 \lambda^2 - (\lambda^2 - \omega^2)^2} \cos \lambda t + \frac{2\kappa \lambda f}{4\kappa^2 \lambda^2 - (\lambda^2 - \omega^2)^2} \sin \lambda t. \end{aligned} \quad (7.38)$$

A и B — произвольные постоянные, находятся из начальных условий.

Исследования переходного процесса установления стационарных вынужденных колебаний, резонанса, биений, возникающих при $\kappa = 0$ и $\lambda \approx \omega_1$ (рис. 7.20-7.22), могут быть, конечно, проведены с использованием формул (7.38) простым табулированием с выводом результатов на экран компьютера в форме, удобной для восприятия; они же могут быть и объектами численного моделирования.

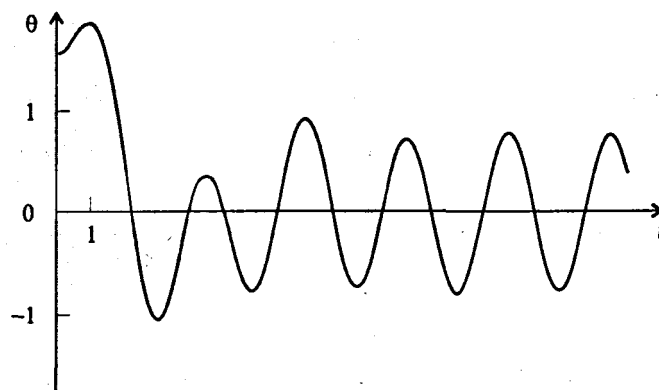


Рис. 7.20. Установление стационарных вынужденных колебаний маятника при наличии трения при $\kappa = 0,5$; $\omega = \pi/2$, $\lambda = \pi$, $f = 2\pi$.

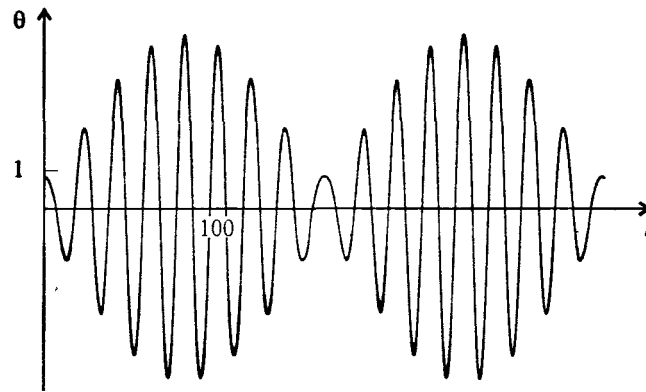


Рис. 7.21. Биения в системе с близкими частотами собственных колебаний и с вынуждающей силой при $k = 0$; $\omega = 889\pi/9000$, $\lambda = \pi/9$, $f = \pi/70$

Возвратимся к уравнению нелинейных вынужденных колебаний (7.36). Его аналитическое решение отсутствует, и возможно лишь численное. Сформулируем ряд задач: как нелинейность влияет (при больших амплитудах движения) на период вынужденных колебаний, на резонанс, на период биений при $\lambda \approx \omega$ и т.д. Однако математики и физики давно убедились в том, что переход от линейного к нелинейному может изменить не только количественные характеристики процесса, но и дать новое качество. В данном случае - возникновение при некоторых условиях хаотического движения маятника. Сама возможность возникновения таких движений в простых динамических системах была обнаружена относительно недавно и поразила воображение многих математиков, физиков, химиков, биологов, в которых ситуации с хаотическими движениями, как оказалось, отнюдь не редкость. Пример такого процесса будет приведен впоследствии.

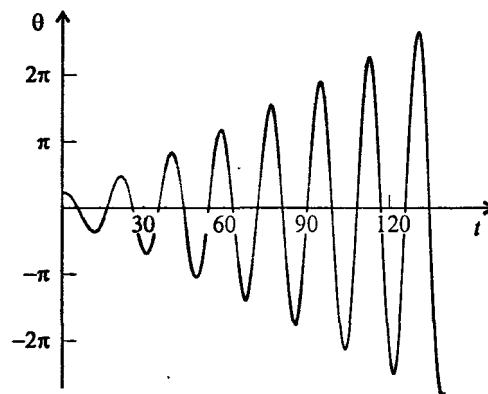


Рис. 7.22. Возрастание амплитуды колебаний при прохождении через резонанс при $k = 0$; $\omega = 889\pi/9000$, $\lambda = \pi/9$, $f = \pi/70$

Параметрические колебания. Рассмотрим еще один вид колебаний маятника, когда на него внешние силы непосредственно не действуют, но внутри системы происходят некоторые события, приводящие к зависимости от времени параметров, входящих в уравнение движения. В этом случае колебательные движения называют параметрическими.

Простейший пример - раскачивание качелей усилиями того человека, который стоит на этих качелях. Все знают, что, приседая, и отталкиваясь «в такт», можно сильно разогнать качели. Указанные приседания сводятся к периодическому изменению центра тяжести системы, или, что почти равносильно, длины нити подвеса. Поскольку длина нити подвеса определяет частоту колебаний, то математическая модель явления – уравнение

$$\frac{d^2\theta}{dt^2} + 2\kappa \frac{d\theta}{dt} + \omega^2(t) \sin \theta = 0, \quad (7.39)$$

где $\omega(t)$ — заданная функция, определяющая закон изменения частоты. Мы ограничимся простейшим случаем гармонического изменения $\omega^2(t)$:

$$\omega^2(t) = \omega_0^2(1 + \alpha \cos \lambda t),$$

где λ - частота изменения величины $\omega^2(t)$.

При малых амплитудах колебаний и отсутствии трения уравнение (7.39) превращается в

$$\frac{d^2\theta}{dt^2} + \omega_0^2(1 + \alpha \cos \lambda t)\theta = 0.$$

Решение любого из этих уравнений возможно лишь численно. Одна из интереснейших особенностей уравнения (7.40) - так называемый, параметрический резонанс - допускает частичное аналитическое исследование, однако слишком сложное, чтобы его здесь приводить. Параметрический резонанс состоит в том, что при некоторых соотношениях частот λ и ω_0 , а именно $\lambda \approx \frac{\omega_0}{2}$, $\lambda \approx \omega_0$, $\lambda \approx \frac{3\omega_0}{2}$, $\lambda \approx 2\omega_0, \dots$ и при определенных значениях величины α в системе возникают нарастающие колебания. На рис. 7.23 схематически изображена фазовая диаграмма системы в переменных $\gamma = \frac{\lambda}{\omega_0}$ и α , на ней заштрихованы зоны параметрического резонанса.

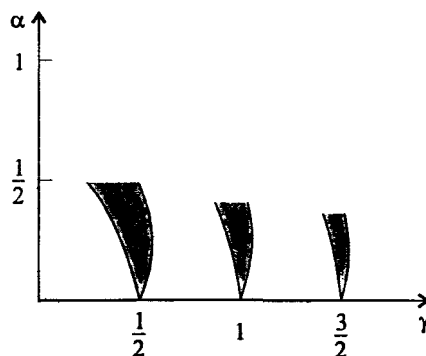


Рис. 7.23 Фазовая диаграмма с зонами параметрического резонанса

Понимать такую фазовую диаграмму надо следующим образом: если значения параметров γ , α принадлежат заштрихованной области, то при них имеет место параметрический резонанс. Очень интересно то, что он наступает скачком при пересечении границы на фазовой плоскости.

Как можно численно установить границу зоны параметрического резонанса, например, первой? - Для этого надо задаться некоторыми значениями α (например, 0,1) и γ (например, 0,3), не принадлежащими зоне неустойчивости, и проинтегрировать численно уравнение (7.40). Удобно предварительно обезразмерить время переменной $\tau = \omega_0 t$, после чего уравнение примет вид

$$\frac{d^2\theta}{d\tau^2} + (1 + \alpha \cos \gamma \tau)\theta = 0. \quad (7.41)$$

Здесь $\gamma = \frac{\lambda}{\omega_0}$. Затем, медленно увеличивая γ (например, с шагом 0,01) и не меняя α , интегрировать уравнение (7.41), пока не попадешь в зону неустойчивости, и далее, пока не выйдешь из нее. Затем следует увеличить α (например, взяв $\alpha = 0,2$) и снова повторить процедуру прохождения по значениям γ и т.д. - постепенно вырисовывается картина границы зоны параметрического резонанса на фазовой плоскости.

Нарастание колебаний при параметрическом резонансе, описываемом уравнением (7.40), является неограниченным. Физически такого быть не может. Ограничение амплитуды колебаний наступает либо за счет учета трения, либо при возврате к $\sin \theta$ в уравнении (7.39), либо за счет обо-

их факторов. Следует учесть, что наличие трения не только ограничивает размах параметрических колебаний, но и «приподнимает» зоны параметрического резонанса над осью γ на фазовой плоскости α, γ , причем в разной мере. Моделирование этого и других явлений при параметрическом резонансе - интересная исследовательская работа.

Многогранность задачи об одномерных колебаниях. Колебания математического маятника одномерны в том смысле, что они описываются одной функцией $\theta(t)$ (хотя они и происходят в двумерном пространстве - плоскости, но жесткий стержень ликвидирует одну из степеней свободы, и в обычных декартовых координатах $x(t), y(t)$ выражаются друг через друга).

Оказывается, что рассмотренные выше уравнения, особенно линейные (т.е. малых колебаний), обладают высокой универсальностью и описывают ряд процессов в механике твердых тел, газов, в электродинамике и т.д. Так, уравнение малых колебаний

$$\frac{d^2x}{dt^2} + 2\kappa \frac{dx}{dt} + \omega_0^2 x = 0 \quad (7.42)$$

описывает указанные ниже и другие системы (при этом в x, κ, ω вкладывается совершенно разный физический смысл):

- математический маятник;
- пружинный маятник, где сила, действующая на тело, определяется законом Гука;
- «физический» маятник-тело, свободно вращающееся около горизонтальной оси;
- крутильный маятник наручных часов - симметричное тело, совершающее колебания около вертикальной оси под действием спиральной пружины;
- ток в колебательном контуре;
- акустический резонатор Гельмгольца, в котором происходят колебания воздуха в колбе с широким горлышком;
- колебания магнитной стрелки компаса.

Таким образом, наше внимание к колебательному движению не является преувеличенным.

Интересно, что при больших амплитудах универсальность колебательных движений нарушается. Так, $\sin\theta$ в уравнении для математического маятника для других движений заменяется другой нелинейной функцией, и всякий раз задачу приходится решать заново и, чаще всего, численно.

3.8. МОДЕЛИРОВАНИЕ ЯВЛЕНИЙ И ПРОЦЕССОВ В ПРИБЛИЖЕНИИ СПЛОШНОЙ СРЕДЫ

Абстрактное понятие «сплошная среда» широчайшим образом используется в науке. Во многих ситуациях жидкости, газы, твердые тела, плазму можно рассматривать как «сплошные», отвлекаясь от их молекулярного и атомарного устройства. Например, при распространении волн в жидкости или газе реальная дискретность этих сред практически не сказывается на свойствах волн, если длина волны много больше характерного межмолекулярного расстояния; при изучении процессов распространения тепла или диффузии тоже до поры-до времени можно «забыть» об атомарном строении вещества и оперировать такими характеристиками как теплоемкость, теплопроводность, скорость диффузии и др., которые можно обсуждать и практически использовать в технике без выяснения их микроскопической природы. Вообще, «макрофизика» может быть очень полезной чисто практически без привлечения «микрофизики», которая стремится докопаться до объяснения природы явлений, исходя из атомарных и еще более «микроскопических» представлений.

В приближении сплошной среды свойства объекта описываются математически с помощью непрерывных функций от координат и времени: $f(\vec{r}, t)$. За каждым «свойством» закрепляется такая функция, и их взаимосвязанный вид дает полное описание среды.

Существующие задачи можно разделить на два класса: статические и динамические. В первом случае значения величин, характеризующих сплошную среду, не зависят от времени, и требуется найти их пространственное распределение. Хорошо известные примеры: как распределено в пространстве значение электрического поля, созданного неподвижным точечным зарядом? как распределены электрическое поле в конденсаторе? поле постоянного магнита? скорости в стацио-

нарно движущемся по трубе потоке жидкости? На рис. 7.24 дан (схематически) ответ на последний вопрос: чем ближе к стенке трубы, тем меньше скорость из-за естественной вязкости жидкости и трения о стенку трубы. Качественно понять указанную закономерность можно, вероятно, без всяких уравнений, но определить профиль скоростей, т.е. форму огибающей векторов скорости без математического моделирования невозможно. Таких задач, представляющих огромный практический интерес, очень много, а связанные с их решением математические проблемы столь сложны, что чаще всего соответствующее математическое моделирование может быть реализовано лишь на компьютере.

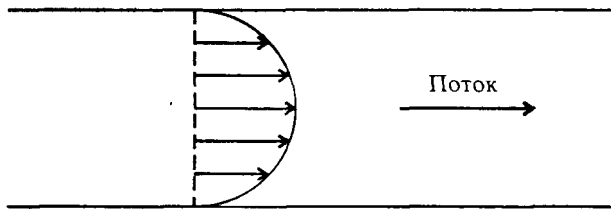


Рис. 7.24. Распределение скоростей в потоке жидкости, движущейся в трубе

Как правило, еще сложнее решение динамических задач. Если электрическое поле создается движущимися зарядами, то определить, как оно меняется во времени в каждой точке пространства - задача очень непростая. Не менее трудно определить эволюцию скорости в разных местах в жидкости, если в некотором месте пульсирует давление; изменения значений температуры в разных точках некоторого тела, которое подогревают изнутри или извне от источников тепла, интенсивность которых изменяется со временем.

Подобные задачи привлекают неослабевающее внимание физиков, научных работников смежных областей, инженеров уже не менее 200 лет. Практическая необходимость в их решении велика; без этого не спроектировать ни современных технических устройств и механизмов, ни строений, ни космических аппаратов, ни многого другого. Главный способ решения таких задач - математическое моделирование. Любопытно, что и сами компьютеры, и входящие в них микро-элементы невозможно спроектировать без оценок электрических полей и потоков тепла от этих устройств.

Поскольку математический аппарат такого моделирования бывает весьма сложен, мы ограничимся лишь двумя относительно простыми задачами, в которых отражается часть общих закономерностей. Одна из них - статическая, другая - динамическая.

Распределение электростатического поля. Что стоит за электрической (кулоновской) силой, заставляющей двигаться заряженную частицу q ? – Ответ хорошо известен: электрическое поле $\vec{E}$, существующее в каждой точке пространства, созданное другими заряженными телами (которые будем считать неподвижными). Если это поле создается одним точечным зарядом Q , то величина напряженности поля зависит от расстояния r от Q до данной точки пространства:

$E = \frac{1}{4\pi\epsilon_0} \cdot \frac{Q}{r^2}$, ее направление - по радиусу от заряда (если Q положителен). Поле это существует

совершенно независимо от «пробного» заряда q и может рассматриваться как сплошная среда. Существуют две взаимосвязанные характеристики электрического поля: напряженность E (вектор-

ная характеристика) и потенциал φ - скалярная. Для поля точечного заряда $\varphi = -\frac{1}{4\pi\epsilon_0} \cdot \frac{Q}{r}$.

Если поле создано не одним, а несколькими зарядами, то напряженность и потенциал в каждой точке можно найти из известного принципа суперпозиции:

$$\vec{E} = \sum_i \vec{E}_i, \quad \varphi = \sum_i \varphi_i$$

где $\vec{E}_i$ и φ_i создаются в этой точке i -м зарядом, рис. 7.25. По отношению к $\vec{E}$ принцип суперпозиции означает необходимость векторного сложения, к φ — «обычного» (с учетом знаков отдельных потенциалов).

Зная потенциал в каждой точке поля, т.е. функцию $\Phi = \varphi(x, y, z)$, можно найти напряженность в каждой точке чисто математическим путем, отражающим тот факт, что проекция вектора

напряженности на любое направление есть скорость изменения потенциала в этом направлении:

$$E_x = -\frac{\partial \varphi}{\partial x}, \quad E_y = -\frac{\partial \varphi}{\partial y}, \quad E_z = -\frac{\partial \varphi}{\partial z}. \quad (7.43)$$

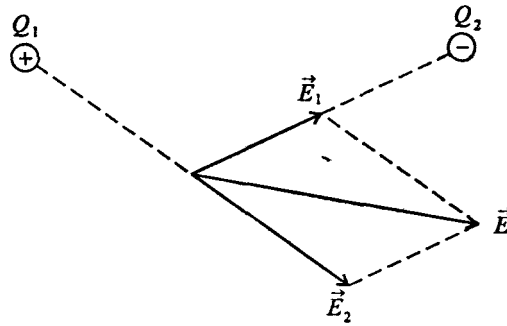


Рис. 7.25. Нахождение напряженности электрического поля по принципу суперпозиции

Частным случаем (7.43) являются формулы $\vec{E} = \frac{1}{4\pi\epsilon_0} \frac{Q}{r^3} \vec{r}$ и $\varphi = -\frac{1}{4\pi\epsilon_0} \frac{Q}{r}$ для одного точечного заряда. Действительно, фиксируем некоторую точку A поля на расстоянии r от заряда Q и введем локальную систему координат с центром в A ; у этой системы ось r является продолжением радиуса-вектора r , а две другие оси x и y - перпендикулярны к ней. Примем, что $\varphi_A = -\frac{1}{4\pi\epsilon_0} \frac{Q}{r}$, и найдем $\vec{E}_A$, опираясь на формулы (7.43). Поскольку φ_A от x и y не зависит, то $\frac{\partial \varphi_A}{\partial x} = 0$, $\frac{\partial \varphi_A}{\partial y} = 0$, а

$$\frac{\partial \varphi_A}{\partial r} = \frac{1}{4\pi\epsilon_0} \frac{\partial}{\partial r} \left(-\frac{Q}{r} \right) = \frac{1}{4\pi\epsilon_0} \frac{Q}{r^2}.$$

таким образом, $\vec{E}_A = \left(0, 0, \frac{1}{4\pi\epsilon_0} \frac{Q}{r^2} \right)$ - т.е. мы пришли к известному результату о величине и направлении поля, созданного точечным зарядом.

Расчет электрического поля - важная в прикладном плане задача. В реальных конструкциях поле создается не одним-двумя точечными зарядами, а достаточно причудливо расположенными в пространстве заряженными телами самых разнообразных форм: пластины, плоские и изогнутые; штыри; правильные и деформированные сфероиды и т.д. Для инженера и научного работника важно иметь наглядную картину поля, изображенного некоторым условным образом. Самое неудобное изображение, почти не используемое - нарисовать много стрелок, соответствующих напряженности поля в разных точках, так, чтобы длины стрелок были пропорциональны напряженностям. Такой рисунок является громоздким, стрелки на нем пересекаются, мелкие детали выявить трудно. Есть два классических способа для наглядного изображения поля: поверхностями (или линиями) равного потенциала и силовыми линиями поля.

Можно доказать, что для любого электростатического поля множество точек, потенциал в которых одинаков, т.е. точек, удовлетворяющих уравнению $\varphi(x, y, z) = \varphi_0$, при любом φ_0 образует замкнутую поверхность (так называемую, эквипотенциальную поверхность). Для одного точечного заряда это сфера; в общем случае эта поверхность может быть очень сложной. Для многих технических приложений знать форму таких поверхностей просто необходимо - например, чтобы, располагая детали конструкции, избежать между ними большой разности потенциалов. Линии равного потенциала являются сечениями поверхности равного потенциала той плоскостью, в которой строится изображение.

Силовые линии, как известно из любого учебника физики, есть такие линии, касательные к которым в каждой точке задают направление вектора напряженности поля. Силовые линии никогда не пересекаются между собой. Они начинаются на положительных зарядах и либо заканчиваются на отрицательных, либо уходят «на бесконечность». По обычному соглашению число сило-

вых линий, исходящих из точечного заряда, пропорционально величине этого заряда; коэффициент пропорциональности выбирается таким, чтобы изображение было легко читаемым.

Обсудим практический метод построения картины поверхностей равного потенциала для системы, состоящей из нескольких точечных зарядов произвольной величины и знака, любым способом расположенных в пространстве. Введем некоторую систему координат, начало которой удобнее расположить в «пустой» точке, т.е. ни на одном из зарядов. Пусть в этой системе координаты зарядов имеют значения $\vec{r}_j = (x_j, y_j, z_j)$, $j = 1, 2, \dots, p$, где p - число зарядов.

Поскольку изображать трехмерные поверхности - дело достаточно сложное, рассмотрим вначале построение линий равного потенциала (изолиний), образованных сечением поверхности равного потенциала некоторой плоскостью; пусть, для определенности, это будет плоскость l' . Воспользуемся методом сеток, играющим в моделировании свойств сплошных сред исключительно важную роль.

Выберем по осям x и y некоторые шаги h_x и h_y и покроем плоскость сеткой, образованной прямыми, параллельными осям x и y и отстоящими друг от друга на расстояниях h_x и h_y соответственно. Точки пересечения этих прямых — узлы сетки. Пронумеруем их так: начало координат $(0, 0)$, следующий по оси x вправо - $(0, 1)$, влево - $(0, -1)$; по оси y вверх - $(1, 0)$, вниз $(-1, 0)$ и т.д. Значения потенциала, создаваемого системой зарядов $Q_1 \dots Q_p$ в узле (i, k) , согласно принципу суперпозиции, таково (обратим внимание, что здесь i - номер строки, k - столбца сетки):

$$\Phi_{ik} = - \sum_{l=1}^p \frac{1}{4\pi\epsilon_0} \cdot \frac{Q_l}{\sqrt{(kh_x^2 + ih_y^2)}}.$$

Ограничимся прямоугольной областью в плоскости xy : $[-mh_x, mh_x]$ по оси x и $[-nh_y, nh_y]$ по оси y . В этой области $(2m + 1) \cdot (2n + 1)$ узлов. Вычислим значения потенциала в каждом из них по указанным формулам; для ЭВМ эта задача совершенно элементарна, даже если m и n составляют несколько десятков или сотен. В результате получим матрицу значений потенциала.

Фиксируем некоторое значение потенциала Φ и построим изолинию, соответствующую этому значению. Для этого проходим, к примеру, по i -ой горизонтальной линии сетки и ищем среди ее узлов такие соседние, значения потенциала в которых «захватывают» Φ между собой; признаком этого может служить выполнение неравенства

$$(\Phi_{ik} - \tilde{\Phi}) \cdot (\Phi_{i,k+1} - \tilde{\Phi}) < 0.$$

Если такая пара узлов найдена, то координату точки, в которой $\Phi = \tilde{\Phi}$, найдем приближенно с помощью линейной интерполяции:

$$x = k \cdot h_x + \frac{\tilde{\Phi} - \Phi_{ik}}{\Phi_{i,k+1} - \Phi_{ik}} \cdot h_x, \quad y = i \cdot h_y. \quad (7.44)$$

Найдя в данной горизонтали все такие точки, переходим к следующей горизонтали, пока не исчерпаем их все. Для этого надо совершить двойной циклический проход: во внешнем цикле перебирать i от $-n$ до $+n$, во внутреннем перебирать k от $-m$ до $+m$.

После этого следует аналогично заняться поиском нужных точек на вертикальных линиях сетки. Детали процедуры очевидны; формулы, аналогичные (7.44), имеют вид:

$$x = k \cdot h_x, \quad y = i \cdot h_y + \frac{\tilde{\Phi} - \Phi_{ik}}{\Phi_{i+1,k} - \Phi_{ik}} \cdot h_y. \quad (7.45)$$

После прохождения всех горизонтальных и вертикальных линий сетки находятся все те точки на этих линиях, в которых потенциал равен $\tilde{\Phi}$. Проведя — мысленно или на бумаге — кривую, плавно проходящую через ближайшие точки, получаем искомую изолинию (разумеется лишь в том случае, если значение $\tilde{\Phi}$ выбрано разумно и такая линия есть в пределах рассматриваемой области). Затем берем другие значения $\tilde{\Phi}$ и повторяем указанную процедуру, получая таким образом семейство изолиний.

Приведенная ниже программа реализует указанные построения. Предполагается, что все заряды лежат в одной плоскости, и изолинии строятся тоже лишь в этой плоскости.

Программа 149. Программа построения линий равного потенциала.

```
Program Potential;
Uses Crt, Graph;
Const N = 100; (Размер сетки NxN)
Var X, Y, Q, G : Array[1..10] Of Real; F : Array[0..N, 0..N] Of Real;
    I, J, M, L, K: Integer; A, B, R: Real;
Begin
  WriteLn('Сколько зарядов? (не более 10)'); ReadLn(K);
  Write('Ведите координаты x, y и величины зарядов q');
  WriteLn('(координаты - в диапазоне 0-1)');
  For I := 1 To K Do
    Begin Write('x[' , I, ']=');
      ReadLn(X[I]); Write('y[' , I, ']='); ReadLn(Y[I]);
      Write('q[' , I, ']= '); ReadLn(Q[I])
    End;
  For I := 0 To N Do
    For J := 0 To N Do
      For M := 1 To K Do
        Begin
          R := Sqrt(Sqr(I / N - X[M]) + Sqr(J / N - Y[M]));
          If R>=1E-6 Then F[I,J]:= F[I,J]+Q[M]/R Else F[I,J]:=1E+8
        End;
      Write('Сколько построить изолиний? (не более 10)'); ReadLn(L);
      WriteLn('Введите значения потенциала g для построения изолиний');
      For I := 1 To L Do
        Begin Write ('g[' , I, ']='); ReadLn(G[I]) End;
  DetectGraph(I, J); InitGraph(I, J, "");
  For I := 1 To K Do
    Begin
      A := X[I] * GetMaxX; B := (1 - Y[I]) * GetMaxY;
      Circle(Round(A), Round(B), 4); FloodFill(Round(A), Round(B),
        GetColor) ;
    End;
  For M := 1 To L Do
    Begin
      B := G[M]; SetColor(M);
      For I := 0 To N Do
        For J := 0 To N - 1 Do
          If (F[I, J] - B) * (F[I, J + 1] - B) < 0
            Then Begin
              A:=(J+(B-F[I,J]) / (F[I,J+1]-F[I,J]))/N;
              Circle(Round(I/N*GetMaxX), Round((1-A)*GetMaxY), 1)
            End;
          For J := 0 To N Do
            For I := 0 To N - 1 Do
              If (F[I, J] - B) * (F[I + 1, J] - B) < 0
                Then Begin
                  A:=(I+(B-F[I,J]) / (F[I+1,J]-F[I,J]))/N;
                  Circle(Round(A*GetMaxX), Round((1-
                    J/N)*GetMaxY), 1)
                End
            End;
          End;
      SetColor(15); OutTextXY(10, 50, 'для продолжения нажмите любую клавишу');
      Repeat Until KeyPressed; CloseGraph;
    End.
```

Несколько примеров использования этой программы приведены на рис. 7.26, 7.27.

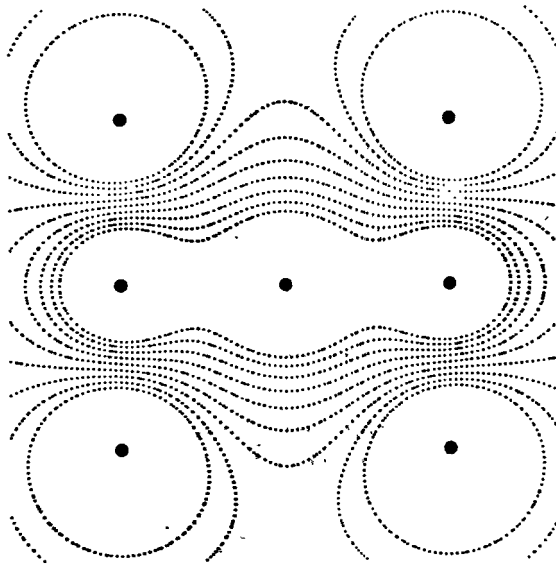


Рис. 7.26. Поле создано семью зарядами $q_1 = q_2 = q_3 = q_4 = 1$, $q_5 = q_6 = q_7 = -1$, имеющими соответственно координаты $(0,2;0,2)$, $(0,8;0,8)$, $(0,2;0,8)$, $(0,8;0,2)$, $(0,2;0,5)$, $(0,5;0,5)$, $(0,8;0,5)$. Изолинии построены для потенциалов -4, -3, -2, -1, 0, 1, 2, 3, 4

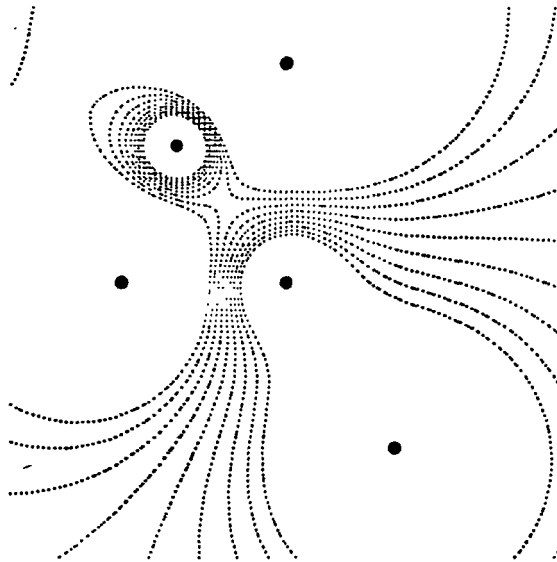


Рис. 7.27. Поле создано пятью зарядами $q_1 = 1$, $q_2 = -2$, $q_3 = 2$, $q_4 = -3$, $q_5 = 1$, имеющими соответственно координаты $(0,3; 0,75)$, $(0,2; 0,5)$, $(0,7; 0,2)$, $(0,5; 0,9)$, $(0,5; 0,5)$. Изолинии построены для потенциалов -4, -3, -2, -1, 0, 1, 2, 3, 4

Оставим технические вопросы на самостоятельное решение и обсудим некоторые принципиальные. Допустим, между двумя ближайшими узлами выполняется записанное выше неравенство - означает ли это, что между ними действительно лежит *одна* точка, в которой $\Phi = \tilde{\Phi}$? Ответить нетрудно: да, если потенциал между этими узлами меняется монотонно. Если же узлы столь редки (т.е. h_x и (или) h_y слишком велики), что потенциал между соседними узлами меняется немонотонно, то числа, полученные по формулам (6.44), (6.45), не имеют практически никакого отношения к реальным точкам, в которых $\Phi = \tilde{\Phi}$; это утверждение проиллюстрировано рис. 7.28.

Очевидно, что для получения изолиний следует брать достаточно малые h_x и h_y . Проверка достоверности (эмпирическая) состоит в том, что строится картина изолиний с некоторыми h_x и h_y (часто берут $h_x = h_y$), а затем с вдвое меньшими значениями; если картины близки, то построение на этом завершается.

Даже если все заряды лежат в одной плоскости (как это было на рис. 7.26 и 7.27), поле существует, конечно, и вне этой плоскости. Один из способов наглядного построения изображения поля - найти изолинии, соответствующие некоторому фиксированному набору значений Φ ; в не-

скольких параллельных плоскостях и представить их на общем рисунке, дающем представление о поверхностях равного потенциала. Для этого программу, приведенную выше, следует слегка дополнить.

Метод сеток в разных задачах физики сплошных сред принимает разное обличие; еще один пример впереди. Однако, во всех случаях за ним скрыта общая идея, обладающая большой познавательной силой - идея дискретизации, т.е. представления непрерывной величины, имеющей бесконечно много значений, отдельными порциями, описываемыми конечным набором значений. Эта идея продуктивна не только в физике, но и в прикладной математике, информатике, других науках.

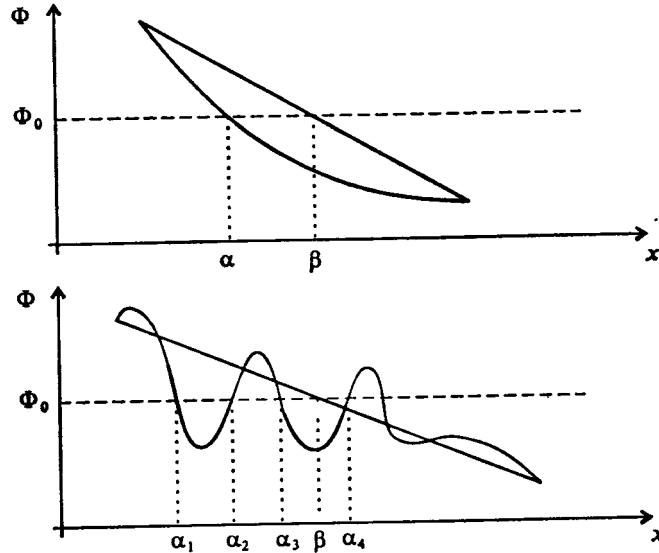


Рис. 7.28 На верхнем рисунке α - точка, в которой $\Phi = \Phi_0$. $\beta \approx \alpha$ - найдено линейной интерполяцией. На нижнем рисунке точек, в которых $\Phi = \Phi_0$, много; β формально найдено линейной интерполяцией

Для построения силовых линии поля можно поступить следующим образом. Выберем некоторую точку с координатами $(\alpha_0, \beta_0, \gamma_0)$ и найдем в ней напряженность поля

$$\vec{E}_0 = (E_x^{(0)}, E_y^{(0)}, E_z^{(0)})$$

по правилу суперпозиции

$$E_x^{(0)} = \sum_{i=1}^n \frac{Q_i (x_i - \alpha_0)}{\Gamma_{i0}^3}, \quad E_y^{(0)} = \sum_{i=1}^n \frac{Q_i (y_i - \beta_0)}{\Gamma_{i0}^3}, \quad E_z^{(0)} = \sum_{i=1}^n \frac{Q_i (z_i - \gamma_0)}{\Gamma_{i0}^3} \quad (7.46)$$

где

$$\Gamma_{i0} = \sqrt{(x_i - \alpha_0)^2 + (y_i - \beta_0)^2 + (z_i - \gamma_0)^2}.$$

Проведем мысленно в точке $(\alpha_0, \beta_0, \gamma_0)$ касательную к $\vec{E}_0$ и возьмем вдоль нее небольшой отрезок длины h , начинающийся в $\vec{E}_0$; координаты конца отрезка

$$\alpha_1 = \alpha_0 + h \cdot \frac{E_x^{(0)}}{|\vec{E}_0|}, \quad \beta_1 = \beta_0 + h \cdot \frac{E_y^{(0)}}{|\vec{E}_0|}, \quad \gamma_1 = \gamma_0 + h \cdot \frac{E_z^{(0)}}{|\vec{E}_0|}. \quad (7.47)$$

Тем самым получаем координаты точки A' , лежащей на касательной к силовой линии (вместо точки A , лежащей на самой линии) Если h мало, то A' близко к A . Далее, отправляясь от A' , найдем по той же схеме следующую точку B' вблизи силовой линии и т.д. Ломаная $OA'B''...$ приблизительно передает силовую линию. Построение целесообразно начать вблизи какого-нибудь положительного заряда (если он есть) и закончить тогда, когда силовая линия подойдет вплотную к отрицательному заряду или уйдет «на бесконечность».

Построение картины силовых линий, дающих представление о поле - дело неформальное, требующее понимания физической сущности. Два семейства взаимно перпендикулярных линий - равного потенциала и силовых - дают весьма наглядную и исчерпывающую характеристику электростатического поля.

Учитывая трудности визуализации трехмерных изображений, целесообразно ограничиться (по крайней мере вначале) рассмотрением ситуаций, когда все заряды лежат в одной плоскости; тогда силовая линия, начинающаяся из любой точки данной плоскости, из этой плоскости не выйдет, и получится легко воспринимаемая картина.

Способ получения формул (7.47) есть частный случай приема линеаризации - сведения сложной зависимости к простейшей линейной для малых расстояний (или времен). Это мощнейший прием в моделировании физических процессов и в построении многих методов численного анализа. Фактически он лежит в основе дифференциального исчисления - само понятие производной возникает при линеаризации функции.

3.9. МОДЕЛИРОВАНИЕ ПРОЦЕССА ТЕПЛОПРОВОДНОСТИ

То, что тела могут проводить тепло, общеизвестно. Если один из концов длинного стержня поместить в костер, то, если стержень сделан не из горючего или легко плавящегося материала, другой конец через некоторое время тоже нагреется; как быстро и насколько - зависит от материала, размеров стержня и других факторов. Процесс теплопроводности - один из, так называемых, процессов переноса, играющих огромную роль в природе и в технике. Другие процессы такого рода - диффузия, благодаря которой смешиваются разные жидкости или газы, процессы гидро- и аэродинамики (т.е. переноса (движения) жидкостей и газов).

Хотя каждый из таких процессов имеет собственные закономерности, между ними много общего. Эти процессы происходят в сплошной среде, о которой шла речь выше; при их математическом моделировании используется один и тот же математический аппарат - дифференциальные уравнения в частных производных.

Ограничимся одной из самых простых задач данного класса - переносом тепла в однородном стержне. Рассмотрим линейный стержень, боковая поверхность которого не проводит тепла (теплоизолирована). Если в начальный момент стержень неравномерно нагрет, то в нем будет происходить перераспределение тепла; при отсутствии внутренних источников тепла его температура, в конце концов, выровняется.

Поскольку стержень линеен и однороден, то распределение температуры в пространстве характеризуется одной координатой x .

Температура (обозначим ее u) зависит от x ; кроме того, она может меняться со временем, т.е. является функцией двух переменных $u(x, t)$. Изменение этой функции вдоль стержня, «скорость» которого определяется производной по x , и изменение ее со временем, скорость которого определяется производной по t , взаимосвязаны и, как будет показано ниже, входят в одно уравнение.

Уравнение теплопроводности. Получим уравнение, описывающее процесс изменения температуры в стержне. Фиксируем некоторую точку x_0 (рис. 7.29) и выделим около нее малый участок стержня длиной Δx . Искомое уравнение есть по существу уравнение теплового баланса (т.е. сохранения энергии): изменение количества тепла в избранном участке стержня за счет притока и (или) оттока его через два сечения приведет к нагреванию или охлаждению этого участка в соответствии с его теплоемкостью. Выразим все это математическим языком.

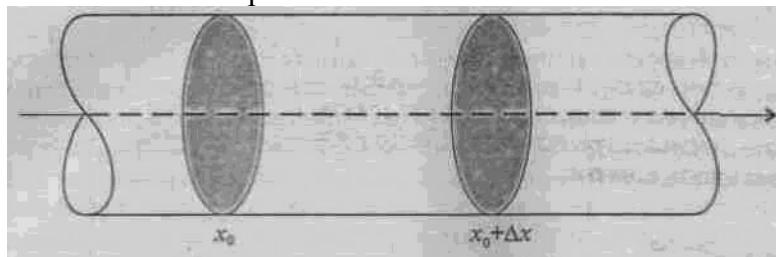


Рис. 7.29. Участок линейного стержня

Количество тепла, проходящее через поперечное сечение стержня в точке x_0 за время Δt , пропорционально площади поперечного сечения S , градиенту температуры $\frac{\partial u}{\partial x}$ и промежутку времени Δt : $\Delta Q \sim S \cdot \frac{\partial u}{\partial x} \cdot \Delta t$, рис. 7.30. Если с S и Δt все очевидно, то появление производной $\frac{\partial u}{\partial x}$

требует пояснения. За ней стоит тот экспериментальный факт, что поток тепла ΔQ , через некоторый участок стержня длиной Δx тем больше, чем больше разность температур ($|u_1| - |u_2|$) на его концах и чем меньше расстояние Δx :

$$\Delta Q \sim \frac{u_1 - u_2}{\Delta x} = \frac{\Delta u}{\Delta x} \rightarrow \frac{\partial u}{\partial x} \text{ при } \Delta x \rightarrow 0.$$

Вводя коэффициент пропорциональности k , называемый коэффициентом теплопроводности, получаем

$$\Delta Q|_{x=x_0} = k \cdot S \cdot \left. \frac{\partial u}{\partial x} \right|_{x=x_0} \cdot \Delta t.$$

Значение k определяется материалом стержня и для нескольких материалов приведено в табл. 7.6 (в единицах системы СИ: $\frac{\text{Вт}}{\text{м} \cdot \text{К}}$).

Таким образом, различия в теплопроводности разных материалов огромны.

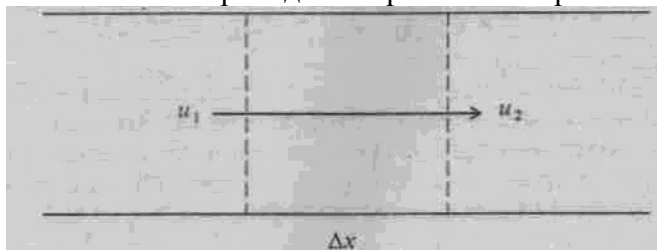


Рис. 7.30. Поток тепла через участок стержня длиной Δx

Теперь запишем количество тепла, проходящее через сечение в точке $x = x_0 + \Delta x$. Оно определяется, естественно, той же формулой:

$$\Delta Q|_{x=x_0+\Delta x} = k \cdot S \cdot \left. \frac{\partial u}{\partial x} \right|_{x=x_0+\Delta x} \cdot \Delta t$$

с условием, что производная $\frac{\partial u}{\partial x}$ берется в точке $x = x_0 + \Delta x$. Для получения искомого уравнения ее надо выразить через значение в точке x_0 .

Таблица 7.6

Значение коэффициента теплопроводности для некоторых материалов

Медь	384	Лед (0° C)	2,23	Асбест	0,4 - 0,8
Алюминий	209	Бетон	0,7 - 0,2	Дерево	0,1 - 0,2
Сталь	47	Кирпич	0,7	Воздух	0,034

Имеем, ограничиваясь первым порядком приращения Δx ,

$$\left. \frac{\partial u}{\partial x} \right|_{x=x_0+\Delta x} \approx \left. \frac{\partial u}{\partial x} \right|_{x=x_0} + \frac{\partial}{\partial x} \left(\left. \frac{\partial u}{\partial x} \right|_{x=x_0} \right) \cdot \Delta x,$$

в силу чего

$$\Delta Q|_{x=x_0+\Delta x} = k \cdot S \cdot \left(\left. \frac{\partial u}{\partial x} \right|_{x=x_0} + \left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_0} \cdot \Delta x \right) \cdot \Delta t.$$

Если через сечения $x = x_0$ и $x = x_0 + \Delta x$ за время Δt прошло разное количество тепла, то та его часть, которая пошла на нагревание (или, в зависимости от знака, на охлаждение) этого участка стержня, есть

$$\Delta Q = \Delta Q|_{x=x_0+\Delta x} - \Delta Q|_{x=x_0} = k \cdot S \cdot \left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_0} \cdot \Delta x \cdot \Delta t.$$

Пусть за то же время температура участка изменилась на Δu ; как известно, это связано с изменением ΔQ соотношением $\Delta Q = mc\Delta u$, где m - масса, c - удельная теплоемкость. Приравняем два выражения для ΔQ :

$$mc\Delta u = k \cdot S \cdot \left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_0} \cdot \Delta x \cdot \Delta t.$$

Поскольку массу можно представить как $m = \rho \cdot S \cdot \Delta x$ (ρ - плотность вещества), то, поделив обе части уравнения на Δt и перейдя к пределу при $\Delta t \rightarrow 0$, получим

$$\rho c \frac{\partial u}{\partial t} = k \cdot \frac{\partial^2 u}{\partial x^2}. \quad (7.48)$$

Это - основное уравнение теплопроводности для однородного стержня. Как следует из процедуры вывода, это уравнение локально, т.е. в данный момент времени и в данной точке выражает закон сохранения энергии.

В уравнение (7.48) входят три постоянные, характеризующие вещество. Удобно объединить их в одну, переписав уравнение в виде

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2}, \quad (7.49)$$

где $a = \sqrt{\frac{k}{c\rho}}$ - так называемый, коэффициент температуропроводности. Обозначение a^2 в (7.49)

удобно, так как фиксирует знак этого коэффициента - он всегда положителен.

Уравнение (7.49) - одно из самых простых дифференциальных уравнений в частных производных. Несмотря на его элементарный вид, решение такого уравнения даже в простейшей ситуации есть весьма сложная задача.

Уравнение теплопроводности в трехмерном случае. Описанный выше вывод уравнения теплопроводности достаточно элементарен. Рассмотрим вывод уравнения теплопроводности в трехмерном случае, используя более общий аппарат математического анализа.

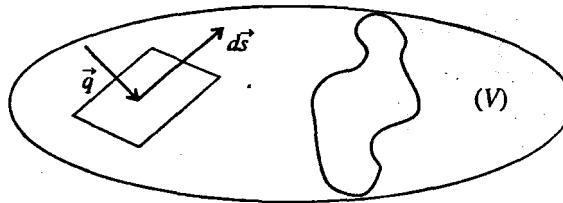


Рис. 7.31. Иллюстрация к выводу уравнения теплопроводности в трехмерном случае

Рассмотрим некоторое тело (V), ограниченное поверхностью (S) (рис. 7.31). Закон сохранения энергии должен выполняться для любой части тела (V). По этому закону скорость изменения энергии в теле равна потоку энергии через его границу. Имеем для энергии в объеме V

$$\iiint_V \varepsilon(\vec{r}, t) dV,$$

где $\varepsilon(\vec{r}, t)$ - объемная плотность энергии.

Поток энергии через границу тела S равен

$$\iint_{(S)} \vec{q} d\vec{S},$$

$\vec{q}$ - поток энергии. В этих формулах фигурируют тройной и поверхностный (первого рода) интегралы. Закон сохранения энергии (интегральный) примет вид

$$\frac{\partial}{\partial t} \iiint_V \varepsilon(\vec{r}, t) dV = - \iint_{(S)} \vec{q} d\vec{S},$$

Применяя к правой части теорему Остроградского-Гаусса, получаем

$$\iiint_V \left(\frac{\partial \varepsilon}{\partial t} + \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} \right) dV = 0.$$

Поскольку это соотношение должно выполняться для любой части тела (V), то необходимо и достаточно, чтобы в любой точке $\vec{r}$ и в любое мгновение t имело место равенство нулю подынтегрального выражения. Учитывая, что плотность энергии $\varepsilon(\vec{r}, t)$ пропорциональна температуре тела, а поток энергии пропорционален градиенту температуры, получаем (опуская детали) уравне-

ние

$$\frac{\partial u}{\partial t} = a^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right), \quad (7.50)$$

где $u = u(\vec{r}, t)$ - температура в точке $\vec{r}$ в момент t . Уравнение (7.50) является трехмерным аналогом уравнения (7.49).

Далее будет продолжено лишь рассмотрение задачи о теплопроводности в стержне.

Начальные и краевые условия. Уравнения (7.49), (7.50) описывают процесс изменения температуры тела (перенос тепла) во времени и в пространстве. Ясно, что для отслеживания такого процесса надо знать распределение температуры в теле в некоторый начальный момент времени:

$$u(x, 0) = u|_{t=0} = f(x), \quad (7.51)$$

где $f(x)$ - заданная функция. Кроме того, в тех местах, где возможен теплообмен с окружающей средой, надо знать условия этого теплообмена. Для стержня с теплоизолированной боковой поверхностью такими местами являются концы. Пусть длина стержня l ; если один конец имеет координату $x = 0$, а другой - $x = l$, то простейший вариант краевых условий - постоянная (но не обязательно одинаковая) температура на каждом конце стержня:

$$u(0, t) = u|_{x=0} = \tilde{u}_0, \quad u(l, t) = u|_{x=l} = \tilde{u}_l.$$

Нижеследующее утверждение физически очевидно, но его строгое математическое доказательство весьма непросто: *дифференциальное уравнение (7.49) при начальном условии (7.51) и краевых условиях (7.52) имеет единственное решение.*

Аналитические методы решения задачи одномерной теплопроводности существуют, но требуют значительной математической подготовки, к тому же решение обычно получается в виде ряда Фурье, и по его виду протекание процесса неочевидно. В двух- и трехмерном случаях аналитическое решение чаще всего получить не удастся (по крайней мере, в практически полезном виде). Как и всюду в этой главе, ниже мы используем простейшие численные методы его решения. Вначале, однако, приведем графические результаты решений простейших задач (заимствованные из книги И.Г.Арамановича и В.И.Левина «Уравнения математической физики», Москва, 1969), способствующие пониманию рассматриваемой проблемы.

Пример 1. В конечном стержне (с теплоизолированной боковой поверхностью) оба торцевых сечения теплоизолированы, а начальная температура распределена по следующему закону:

$$f(x) = \begin{cases} u_0, & \text{если } 0 < x < \frac{1}{2}l, \\ 0, & \text{если } \frac{1}{2}l < x < l. \end{cases}$$

Графики температуры построены в некоторые последовательные моменты времени, рис.

7.32. При любом $t > 0$ график симметричен относительно точки $\left(\frac{l}{2}, \frac{u_0}{2}\right)$.

Теплоизоляция концов стержня находит свое выражение в том, что кривые распределения температуры имеют горизонтальные касательные при $x = 0$ и $x = l$. Из физических соображений ясно, что при $t \rightarrow \infty u \rightarrow u_0/2$.

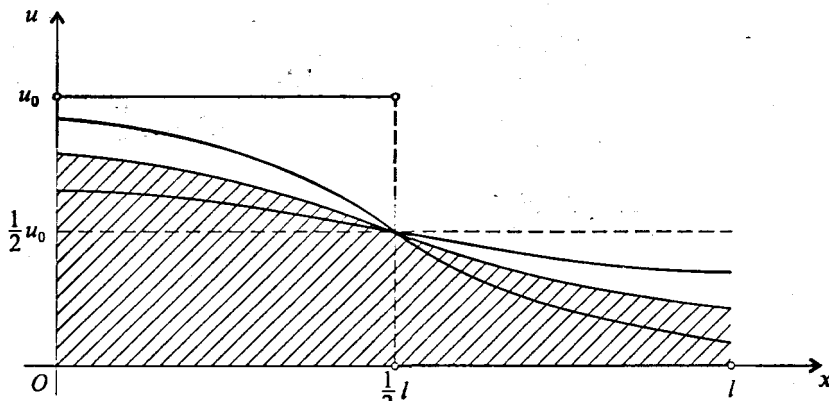


Рис. 7.32. Графическая иллюстрация решения задачи из примера 1

Пример 2. В конечном стержне (с теплоизолированной боковой поверхностью) оба торцевых сечения теплоизолированы, а начальная температура распределена по следующему закону:

$$f(x) = \begin{cases} \frac{2u_0}{l}x, & \text{если } 0 < x \leq \frac{1}{2}l, \\ \frac{2u_0}{l}(l-x), & \text{если } \frac{1}{2}l \leq x < l. \end{cases}$$

Здесь u_0 - максимальное значение температуры.

В точках $\frac{1}{4}l$ и $\frac{3}{4}l$ $u = \frac{1}{2}u_0$ для любого $t > 0$, рис. 7.33. Кроме того, при каждом фиксированном t график u симметричен относительно прямой $x = \frac{1}{2}l$ и каждая его половина симметрична относительно, соответственно, точек $\left(\frac{1}{4}l, \frac{1}{2}u_0\right)$ и $\left(\frac{3}{4}l, \frac{1}{2}u_0\right)$.

Постоянная температура на торцах стержня - простейшее краевое условие. Возможна, однако, и ситуация, когда через торцы происходит теплообмен с окружающей средой. Этот теплообмен, как было установлено Ньютоном, удовлетворяет правилу: поток тепла через единицу поверхности в единицу времени пропорционален разности температур тела и окружающей среды: $\Delta Q = h(u - \tilde{u})$ где u - температура конца стержня, $\tilde{u}$ - температура окружающей среды, h - коэффициент теплообмена. По определению $h > 0$, т.е. $\Delta Q > 0$ соответствует уходу тепла из стержня, $\Delta Q < 0$ - приходу из окружающей среды.

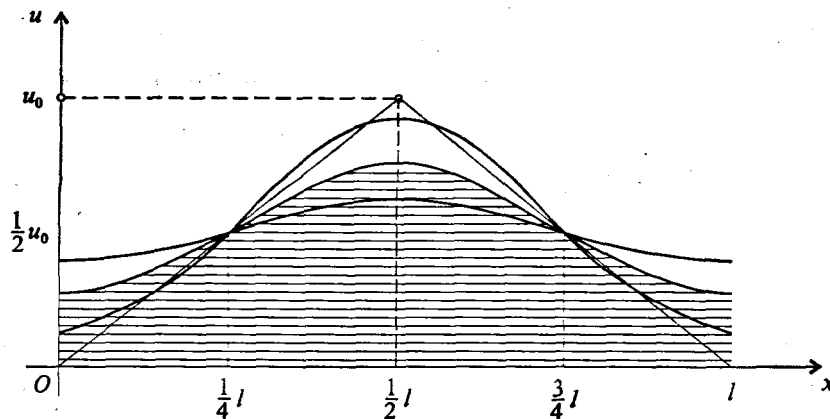


Рис. 7.33. Графическая иллюстрация решения задачи из примера 2

Поскольку поток тепла во внешнюю среду пропорционален градиенту изменения температуры на торце стержня, закон сохранения энергии принимает вид

$$k \frac{\partial u}{\partial x} \Big|_{x=0} = h(u|_{x=0} - \tilde{u}_0), \quad -k \frac{\partial u}{\partial x} \Big|_{x=l} = h(u|_{x=l} - \tilde{u}_l) \quad (7.53)$$

(знак «минус» во второй формуле связан с соотношением направления потока и оси x), k - коэффициент теплопроводности.

Ниже приведен пример эволюции температуры в стержне, у которого один из концов теплоизолирован, а на другом - поддерживается постоянная температура.

Пример 3. В стержне (с теплоизолированной боковой поверхностью) левый конец теплоизолирован: $\frac{\partial u}{\partial x} \Big|_{x=0} = 0$, на правом - поддерживается постоянная температура $u|_{x=l} = \tilde{u}_l$, а начальная температура постоянна по стержню: $u|_{t=0} = u_0$, рис. 7.34.

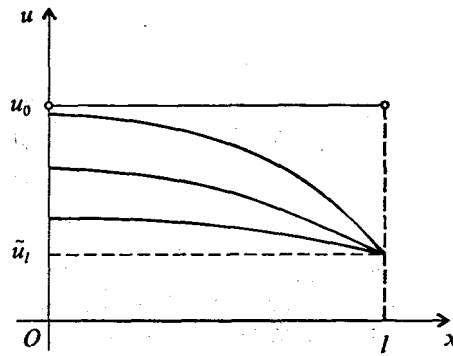


Рис. 7.34. Графическая иллюстрация решения задачи из примера 3

Методы конечных разностей в моделировании свойств сплошных сред. Покажем на примере уравнения теплопроводности наиболее распространенные методы численного интегрирования уравнений в частных производных. В их основе лежит прием дискретизации.

Покроем отрезок $[a, b]$ одномерной сеткой (т.е. разобьем на n равных частей, рис. 7.35) с узлами в точках

$$x_0 = 0, x_1 = x_0 + \Delta x, \dots, x_i = x_0 + i \cdot \Delta x, \dots, x_n = x_0 + n \cdot \Delta x = l.$$

Искомую функцию $u(x)$ будем аппроксимировать ее значениями в узлах сетки. Конечно, такое представление не дает полного описания, но в промежуточных точках, если сетка достаточно «мелкая», возможна интерполяция.

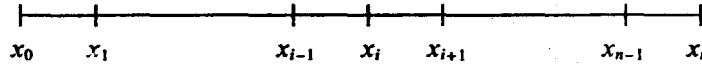


Рис. 7.35. Одномерная сетка

Остановимся на разностной аппроксимации производных. Производная дает информацию о локальном изменении функции в пространстве и, соответственно, связывает ее значения в соседних узлах сетки. Очевидная аппроксимация первой производной в точке x , имеет вид

$$\left. \frac{\partial u}{\partial x} \right|_{x=x_i} \approx \frac{u(x_{i+1}) - u(x_{i-1}))}{2\Delta x}. \quad (7.54)$$

Для крайних точек, однако, такая аппроксимация невозможна, и простейший способ - ограничиться односторонними разностями:

$$\left. \frac{\partial u}{\partial x} \right|_{x=x_0} \approx \frac{u(x_1) - u(x_0)}{\Delta x}, \quad \left. \frac{\partial u}{\partial x} \right|_{x=x_n} \approx \frac{u(x_n) - u(x_{n-1}))}{\Delta x}. \quad (7.55)$$

Разумеется, (7.54) и (7.55) дают простейшие аппроксимации. Втягивая большое количество узлов, можно получить аппроксимации более высокого порядка, но часто бывает достаточно описанных выше. Аналогичная им аппроксимация вторых производных имеет вид

$$\begin{aligned} \left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_i} &\approx \frac{1}{2\Delta x} \left[\left. \frac{\partial u}{\partial x} \right|_{x=x_{i+1}} - \left. \frac{\partial u}{\partial x} \right|_{x=x_{i-1}} \right] = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1}))}{(\Delta x)^2}, \\ \left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_0} &\approx \frac{1}{\Delta x} \left[\left. \frac{\partial u}{\partial x} \right|_{x=x_1} - \left. \frac{\partial u}{\partial x} \right|_{x=x_0} \right] = \frac{1}{\Delta x} \left[\frac{u(x_2) - u(x_0)}{2\Delta x} - \frac{u(x_1) - u(x_0)}{\Delta x} \right] = \\ &= \frac{u(x_2) - 2u(x_1) + u(x_0)}{2(\Delta x)^2}, \\ \left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_n} &\approx \frac{1}{\Delta x} \left[\left. \frac{\partial u}{\partial x} \right|_{x=x_n} - \left. \frac{\partial u}{\partial x} \right|_{x=x_{n-1}} \right] = \frac{u(x_n) - 2u(x_{n-1}) + u(x_{n-2}))}{2(\Delta x)^2}. \end{aligned} \quad (7.56)$$

Что же касается методов интегрирования по времени, то это те же методы, что и для обыкновенных дифференциальных уравнений: Эйлера, Рунге - Кутты и т.д. Так как им тоже свойственна дискретизация, то возникает еще одна, временная сетка. При интегрировании уравнений по времени мы движемся по отдельным слоям, а в каждом слое определяем значение искомой функции на пространственной сетке. Если для интегрирования по времени используется метод Эйлера

или другой одношаговый метод, то для работы со следующим временным слоем используются значения искомой функции из предыдущего слоя, для более сложных - из нескольких предыдущих слоев.

Далее будем индексы, соответствующие временной сетке, писать надстрочно (вверху), а пространственной - подстрочно (внизу). Таким образом, для одномерного уравнения запись u_j^i означает значение функции $u(x, t)$ в j -м временном слое и в i -м узле пространственной сетки. Вернемся к одномерному уравнению теплопроводности (7.49) и сформулируем простейшую возможную схему его интегрирования - явную схему первого порядка - по времени, используя метод Эйлера, по пространству, используя простейшие аппроксимации (7.56). Шаг по времени обозначим Δt , по координате - Δx . Величина $u_i^{k+1} = u(t_{k+1}, x_i)$ находится из разностного уравнения

$$u_i^{k+1} = u_i^k + \frac{a^2 \cdot \Delta t}{(\Delta x)^2} (u_{i+1}^k - 2u_i^k + u_{i-1}^k) \quad (7.57)$$

($k = 0, 1, \dots; i = 1, 2, \dots, n - 1$) для внутренних узлов пространственной сетки; в силу начального условия (7.51)

$$u_i^{(0)} = f(x),$$

где функция $f(x)$ задана и определяет значение температуры при $t = 0$. Что касается значений u_0^k и u_n^k (на концах стержня), то они зависят от типа краевого условия; для случая, когда концы стержня поддерживаются при постоянной температуре, имеем $u_0^k = \tilde{u}_0$, $u_n^k = \tilde{u}_1$, где $\tilde{u}_0, \tilde{u}_1$ - заданные числа.

Теперь остановимся на вопросе об устойчивости и эффективности обсуждаемого метода. Устойчивость понимается в том же смысле, что и для обыкновенных дифференциальных уравнений, но шансов получить неустойчивый метод здесь гораздо больше. Существуют разностные схемы абсолютно неустойчивые, абсолютно устойчивые и условно устойчивые. Первые при любых, сколь угодно малых, шагах так «раскачивают» начальную погрешность, что приводят к результатам, не имеющим ничего общего с реальностью. Вторые ни при каких шагах не «раскачиваются», хотя, конечно, чем меньше шаг, тем меньше разница между приближенным и точным решениями. Третьи устойчивы при одних комбинациях значений Δx и Δt и неустойчивы при других. Исследование, которого мы проводить не будем, показывает, что разностная схема (7.57) устойчива при

$$\frac{a^2 \cdot \Delta t}{(\Delta x)^2} \leq \frac{1}{2}$$

и неустойчива в противном случае.

Эффективность схемы можно представить лишь при сопоставлении с другой схемой того же назначения. Прежде всего, под эффективностью понимают возможность относительно быстро получить решение с достаточной точностью. Иногда оказывается не менее важным объем оперативной памяти под массивы, хранение которых неизбежно в данном методе. Схема (7.57) с точки зрения быстродействия малоэффективна, с точки зрения объема памяти - вполне удовлетворительна, так как, получив значения u_i^{k+1} на некотором временном слое, не обязательно сохранять в ОЗУ значения на предыдущем слое (их можно вывести на диск или на печать).

Получим более эффективный и устойчивый метод. Он аналогичен переходу от метода Эйлера к одному из вариантов метода Рунге - Кутты второго порядка (называемому иногда модифицированным методом Эйлера). Усредним пространственный член уравнения (7.49) по времени:

$$u_i^{k+1} = u_i^k + \frac{a^2 \cdot \Delta t}{(\Delta x)^2} (u_{i+1}^{k+1} - 2u_i^{k+1} + u_{i-1}^{k+1}) + \frac{a^2 \cdot \Delta t}{2(\Delta x)^2} (u_{i+1}^k - 2u_i^k + u_{i-1}^k). \quad (7.58)$$

Это, безусловно, лучше чем в (7.57) аппроксимация производной $\frac{\partial u}{\partial t}$. Исследование показывает, что схема (7.58) (называемая в литературе схемой Кранка-Николсона) абсолютно устойчива и более эффективна.

Расплатой за эффективность является то, что (7.58) - неявная схема, т.е. не формула для

непосредственного расчета, как (7.57), а система линейных алгебраических уравнений для величин $u_1^{k+1}, u_2^{k+1}, \dots, u_{n-1}^{k+1}$ которую еще предстоит решать (поскольку неизвестные на $(k + 1)$ -м временном слое величины u_i^{k+1} входят и в левую, и в правую часть (7.58)). Поскольку неявные схемы, как правило, устойчивей, к ним прибегают часто.

Заметим, что (7.58) есть система специального вида - с трехдиагональной матрицей. В самом деле, если выписать первое, последнее и некоторое промежуточное i -е уравнения, перенося неизвестные в левые части, получим

$$\begin{cases} \left(1 + \frac{a^2 \cdot \Delta t}{(\Delta x)^2}\right) u_1^{k+1} - \frac{a^2 \cdot \Delta t}{2(\Delta x)^2} u_2^{k+1} = u_1^k + \frac{a^2 \cdot \Delta t}{2(\Delta x)^2} (u_2^k - 2u_1^k + u_0^k + u_0^{k+1}), \\ -\frac{a^2 \cdot \Delta t}{2(\Delta x)^2} u_{i-1}^{k+1} + \left(1 + \frac{a^2 \cdot \Delta t}{(\Delta x)^2}\right) u_i^{k+1} - \frac{a^2 \cdot \Delta t}{2(\Delta x)^2} u_{i+1}^{k+1} = u_i^k + \frac{a^2 \cdot \Delta t}{2(\Delta x)^2} (u_{i+1}^k - 2u_i^k + u_{i-1}^k), \\ -\frac{a^2 \cdot \Delta t}{2(\Delta x)^2} u_{n-2}^{k+1} + \left(1 + \frac{a^2 \cdot \Delta t}{(\Delta x)^2}\right) u_{n-1}^{k+1} = u_{n-1}^k + \frac{a^2 \cdot \Delta t}{2(\Delta x)^2} (u_n^k - 2u_{n-1}^k + u_{n-2}^k + u_n^{k+1}). \end{cases} \quad (7.59)$$

Конечно, к таким системам можно применять стандартные методы решения систем линейных алгебраических уравнений, но для них существует и специализированный высокоэффективный метод, называемый «методом прогонки». За деталями отсылаем к учебникам по численным методам.

Пример. Рассмотрим динамику изменения температуры в стержне длиной 4 м с теплоизолированными концами, температура на которых поддерживается постоянной и равна 3°C с начальным условием $f(x) = -0,5x^2 + 2x + 3$. Коэффициент a в уравнении (7.49) примем равным 0,78 (выбор этот достаточно произволен).

Для демонстрации работы явной схемы (7.57) произведем расчеты по этой формуле на первом шаге. Ограничимся пятью узлами на пространственной сетке. В начальный момент ($t = 0$) имеем $u_0^{(0)} = 3,0000$, $u_1^{(0)} = 4,5000$, $u_2^{(0)} = 5,0000$, $u_3^{(0)} = 4,5000$, $u_4^{(0)} = 3,0000$.

Из краевых условий получаем $u_0^{(1)} = u_4^{(1)} = 3,0000$. Подставляя в формулу (7.57) соответствующие значения, получаем

$$u_1^{(1)} = u_1^{(0)} + \frac{a^2 \cdot \Delta t}{(\Delta x)^2} (u_2^{(0)} - 2u_1^{(0)} + u_0^{(0)}) = 4,5 + 0,78^2 \cdot (5 - 2 \cdot 4,5 + 3) = 3,8916,$$

$$u_2^{(1)} = u_2^{(0)} + \frac{a^2 \cdot \Delta t}{(\Delta x)^2} (u_3^{(0)} - 2u_2^{(0)} + u_1^{(0)}) = 5,0 + 0,78^2 \cdot (4,5 - 2 \cdot 5 + 4,5) = 4,3916,$$

аналогично получаем $u_3^{(1)} = 3,8916$.

Таблица 7.7

Результаты моделирования процесса теплопроводности, полученные по неявной схеме (7.59)

$x \backslash t$	0	1	2	3	4
0	3,000	4,500	5,000	4,500	3,000
1	3,000	4,000	4,428	4,000	3,000
2	3,000	3,688	3,975	3,688	3,000
3	3,000	3,476	3,669	3,476	3,000
4	3,000	3,325	3,461	3,325	3,000
5	3,000	3,225	3,316	3,225	3,000
6	3,000	3,154	3,218	3,154	3,000
7	3,000	3,106	3,150	3,106	3,000
8	3,000	3,073	3,103	3,073	3,000
9	3,000	3,050	3,071	3,050	3,000
10	3,000	3,034	3,049	3,034	3,000

На рис. 7.36 представлена графическая иллюстрация результатов расчетов.

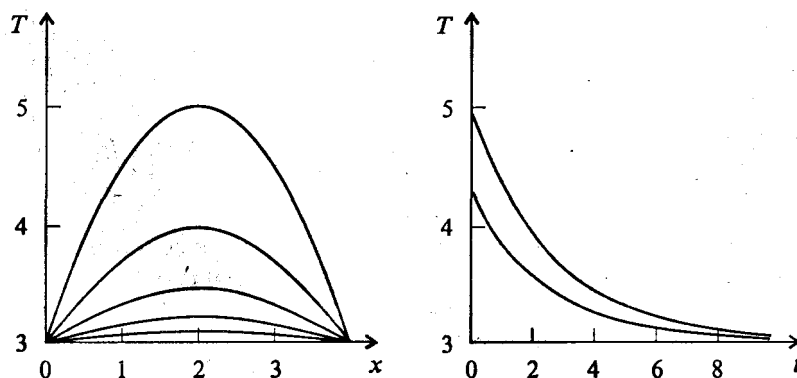


Рис. 7.36. Графики зависимости температуры от координаты в разные моменты времени (сверху вниз $t = 0, t = 2, t = 4, t = 6, t = 8$), в начальный момент времени температура самая высокая, затем она постепенно выравнивается, и зависимости температуры от времени в разных точках стержня. Верхняя кривая соответствует $x = 2$; ниже - $x = 1$ и $x = 3$; прямая линия, совпадающая здесь с осью абсцисс, - значение температуры на концах стержня

Ясно, что по мере эволюции во времени температура стержня будет выравниваться и асимптотически стремиться к 3°C во всех точках.

Контрольные вопросы и задания

1. Какие причины обуславливают особую значимость компьютерного моделирования в физике?
2. Какие аналогии проводятся между реальным и компьютерным экспериментами?
3. Почему при исследовании реальных процессов движения тел нужна дифференциальная форма законов Ньютона?
4. Как зависит сила сопротивления от скорости движущегося тела?
5. Какая из составляющих силы сопротивления - линейная или квадратичная - будет доминировать при погружении в воду полого стального шара - батискафа диаметром 2 м и с толщиной стенки 1 см при достижении им постоянной скорости погружения?
6. Почему учет силы сопротивления среды делает многие, известные из школьного курса физики модели, более реалистичными? Приведите примеры таких моделей.
7. Как надо преобразовать формулировку содержательной задачи, прежде чем приступить к ее решению?
8. Как можно отобразить результаты моделирования в задаче о свободном падении тела в наиболее удобной для восприятия форме?
9. В чем преимущества и недостатки моделирования с помощью составления программ и с использованием табличных процессоров?
10. Разработайте программу для ЭВМ, используя один из методов численного интегрирования системы дифференциальных уравнений, позволяющую моделировать падение тела с учетом сопротивления среды. Предусмотрите интерактивный интерфейс для ввода данных, выбора формы представления результатов и т.д.
Решите с помощью этой программы одну из следующих задач:
 - а) с высоты H падает предмет, через время t он оказывается на земле, требуется определить, с какой скоростью приземлится предмет;
 - б) металлический шарик падает в воде и в глицерине, провести сравнение результатов моделирования;
 - в) определить момент встречи (высоту и время) тела массы m_1 свободно падающего с высоты H_0 , и тела массы m_2 , брошенного вертикально вверх с достаточно большой начальной скоростью.
11. Какова траектория движения тела, брошенного под углом к горизонту, при отсутствии сопротивления среды? Как меняется эта траектория качественно при наличии сильного сопротивления?

12. Для чего производится обезразмеривание величин, характеризующих движение? Возможен ли в рассматриваемой задаче другой способ обезразмеривания?

13. Сделайте сравнительный анализ характеристик движения тела, брошенного под углом к горизонту, с учетом и без учета сопротивления воздуха. Как они будут изменяться с увеличением начальной скорости?

14. Разработайте программы решения задач:

а) при построении модели полета тела, брошенного под углом к горизонту, поверхность Земли считалась плоской, учтите в математической модели кривизну Земли, проведите соответствующее моделирование.

б) произведите моделирование полета тела, брошенного под углом к горизонту на Луне, проведите сравнение с результатами моделирования для Земли при аналогичных начальных условиях;

в) задача о подводной охоте: на расстоянии m под углом α подводный охотник видит неподвижную акулу, на сколько метров выше ее надо целиться, чтобы гарпун попал в цель? как будет выглядеть постановка и решение этой задачи, если акула движется? произведите соответствующее моделирование.

15. В чем могут заключаться усовершенствования приведенной выше модели взлета ракеты?

16. Насколько в действительности хороша аппроксимация, принятая для зависимости силы сопротивления от скорости, при очень больших скоростях?

17. Найдите в специальной литературе данные о характере зависимости силы сопротивления от скорости движения при скоростях порядка скорости звука и больших и внесите усовершенствования в модель.

18. Запишите математическую модель для движения двухступенчатой ракеты.

19. Проведите исследование на тему: с каким минимальным запасом топлива некоторая ракета может вывести на орбиту спутник? Все необходимые параметры задайте правдоподобными самостоятельно.

20. Какой может быть траектория космического аппарата, запускаемого с Земли, относительно нее, если пренебречь влиянием других небесных тел? Чем определяется эта траектория?

21. Как будут выглядеть уравнения движения в системе Земля - Луна - малое небесное тело, если пренебречь влиянием Солнца и других планет?

22. Составьте программу моделирования движения малого космического тела. Получите с помощью этой программы круговую орбиту. Экспериментально подберите безразмерные начальные условия для получения всех видов орбит: эллиптических, параболических, гиперболических. Для эллиптических орбит вычислите длину большой полуоси, эксцентриситет, период обращения.

23. Проверьте в ходе моделирования второй закон Кеплера для эллиптических орбит.

24. Проверьте в ходе моделирования третий закон Кеплера для эллиптических орбит.

25. Уточните модель, учитывая действие на спутник, движущийся вокруг Земли, помимо силы притяжения Земли, слабой постоянной силы W , обусловленной «солнечным ветром».

26. Есть ли качественные различия в задачах о взаимном движении двух небесных тел и двух заряженных частиц, и чем они обусловлены?

27. Произведите моделирование движения тела массы m , несущего заряд q , под действием электростатических сил, создаваемых произвольно расположенной группой тел с зарядами $Q_1, Q_2, \dots, Q_n$ (все они - в одной плоскости).

28. Как выглядит первая нелинейная поправка при переходе от полного уравнения свободных колебаний к уравнению малых колебаний?

29. Какое периодическое движение называют гармоническим?

30. Как выглядит в общем случае формула гармонического разложения периодической функции (разложения в ряд Фурье)?

31. Какие качественные изменения вносит учет трения при анализе движения маятника?

32. С какой частотой происходят вынужденные колебания при наличии гармонической вынуждающей силы?

33. В чем состоит особенность параметрического возбуждения колебательного движения?

34. Изучите в ходе компьютерного моделирования зависимость периода колебаний математического маятника от их амплитуды. Изобразите эту зависимость графически в диапазоне ампли-

туд $0 < \theta < \pi$. Выполните спектральное разложение колебаний для амплитуд $\theta_0 = \frac{3\pi}{4}$, $0,9\pi$, выделив 3-5 гармоник.

35. Изучите с помощью компьютерного моделирования колебания пружинного маятника, движущегося под влиянием упругой силы $F = -ax - bx^3$, где x - смещение из положения равновесия.

Слагаемое $(-ax)$ связано с законом Гука и доминирует при малых x (область упругих деформаций), слагаемое $(-bx^3)$ - нелинейный член силы упругости, доминирующий при больших x . Изучение может включать те же элементы, которые описаны для математического маятника.

36. «Постоянной времени» τ_0 колебательной системы с затуханием называется промежуток времени, за который начальная амплитуда уменьшится в e раз. Для линейной системы она равна $\tau_0 = \frac{1}{k}$. Определите с помощью компьютерного моделирования постоянную времени для затухающих нелинейных колебаний, ее зависимость от начальной амплитуды.

37. Относительно каких процессов атмосферу можно рассматривать как сплошную газовую среду и относительно каких - нельзя?

38. Какие примеры сплошных сред и соответствующих процессов вам известны?

39. Как в общем случае связаны потенциал и напряженность электростатического поля?

40. Что такое эквипотенциальная поверхность? силовая линия?

41. Какие изменения и дополнения следует внести в приведенную выше программу, чтобы она позволила строить трехмерные эквипотенциальные поверхности? их сечения произвольными плоскостями?

42. Реализуйте программу построения силовых линий электростатического поля, создаваемого системой точечных зарядов.

43. Разработайте компьютерную модель, позволяющую строить изолинии поля, создаваемого совокупностью заряженных пластин и точечных зарядов. Создайте с ее помощью изображение

а) поля в плоском конденсаторе;

б) поля, создаваемого пластинами, стоящими под углом друг к другу.

44. В чем заключается процесс теплопроводности, и какие физические механизмы его поддерживают на молекулярном уровне?

45. Как выглядит уравнение теплопроводности в двумерном случае?

46. В чем заключаются начальные и краевые условия в задаче теплопроводности?

47. Как выглядит конечно-разностная аппроксимация первой производной по времени? по пространству? В чем различие этих аппроксимаций для внутренних и граничных узлов сетки?

48. В чем заключаются устойчивость и эффективность численного метода решения краевых задач?

49. В чем состоит принципиальная разница между явной и неявной схемами конечно-разностного решения дифференциальных уравнений?

50. Получите результаты, приведенные выше в примерах моделирования процесса теплопроводности, постройте соответствующие графики. Как еще можно представить результаты расчетов?

51. Выясните, как в рассмотренных примерах моделирования процесса теплопроводности будут изменяться результаты расчетов при уменьшении (увеличении) параметра a .

52. Какими величинами можно обезразмерить переменные в рассмотренных выше примерах моделирования процесса теплопроводности? Проведите обезразмеривание в одном из них для явной и неявной схем.

53. Проведите моделирование теплопроводности, когда начальные условия заданы функцией

$$f(x) = \begin{cases} a_1x + b_1, & \text{если } 0 \leq x < x^*, \\ a_2x + b_2, & \text{если } x^* \leq x \leq l, \end{cases}$$

где x^* - некоторая точка стержня.

§ 4. КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В ЭКОЛОГИИ

4.1. ЭКОЛОГИЯ И МОДЕЛИРОВАНИЕ

Экология - одно из слов, появившихся сравнительно недавно у всех на устах и на страницах газет и журналов. Еще в 60-х годах нашего столетия почти никто, кроме узких специалистов, его не знал, да и большинство из тех, кто знал, использовал в таком смысле, который вряд ли способен заинтересовать широкую общественность. А между тем, термину более 120 лет.

В 1869 г. немецкий естествоиспытатель Эрнст Геккель предложил составной термин «экология» («эко» - дом, жилище, местопребывание и «логос» - наука, знание) как название раздела биологии, ставшего самостоятельным. Классическая экология - наука о взаимодействии организмов и окружающей среды. Сегодня, говоря об экологии, чаще всего имеют в виду не классическую, а, так называемую, социальную экологию, оформившуюся как научное направление и направление общественно-политической деятельности на 100 лет позднее, и занимающуюся проблемами охраны окружающей среды, взаимодействием с ней человеческого сообщества.

В данной главе мы ограничимся некоторыми классическими моделями «старой» экологии, что обусловлено следующими причинами. Во-первых, они достаточно просты и изучены, постановка их вполне очевидна и в познавательном плане интересна и полезна. Во-вторых, модели распространения загрязнений окружающей среды требуют использования весьма сложного математического аппарата, да и сами еще не вполне устоялись. Проблемы охраны окружающей среды чрезвычайно важны, но их обсуждение выходит за пределы нашего курса. Однако, для того, чтобы дать представление о задачах, стоящих перед современными исследователями в этой области, в следующем параграфе приведено описание одной из глобальных моделей, пытающихся выяснить пути взаимодействия экосистемы планеты с индустриальной и экономической системами современного общества.

Остановимся на некоторых понятиях, которые будут встречаться в этой главе. Под *особью* понимается отдельный индивидуум, отдельный организм. *Популяция* - это совокупность особей одного вида, существующих в одно и то же время и занимающих определенную территорию. И, наконец, *сообщество* - это совокупность совместно сосуществующих популяций.

В классической экологии рассматриваются взаимодействия нескольких типов:

- взаимодействие организма и окружающей среды;
- взаимодействие особей внутри популяции;
- взаимодействие между особями разных видов (между популяциями).

Математические модели в экологии используются практически с момента возникновения этой науки. И, хотя поведение организмов в живой природе гораздо труднее адекватно описать средствами математики, чем самые сложные физические процессы, модели помогают установить некоторые закономерности и общие тенденции развития отдельных популяций, а также сообществ. Кажется удивительным, что люди, занимающиеся живой природой, воссоздают ее в искусственной математической форме, но есть веские причины, которые стимулируют эти занятия. Вот некоторые цели создания математических моделей в классической экологии.

1. Модели помогают выделить суть или объединить и выразить с помощью нескольких параметров важные разрозненные свойства большого числа уникальных наблюдений, что облегчает экологу анализ рассматриваемого процесса или проблемы.

2. Модели выступают в качестве «общего языка», с помощью которого может быть описано каждое уникальное явление, и относительные свойства таких явлений становятся более понятными.

3. Модель может служить образцом «идеального объекта» или идеализированного поведения, при сравнении с которым можно оценивать и измерять реальные объекты и процессы.

4. Модели действительно могут пролить свет на реальный мир, несовершенными имитациями которого они являются.

При построении моделей в математической экологии используется опыт математического моделирования механических и физических систем, однако с учетом специфических особенностей биологических систем:

- сложности внутреннего строения каждой особи;
- зависимости условий жизнедеятельности организмов от многих факторов внешней среды;
- незамкнутости экологических систем;

- огромного диапазона внешних характеристик, при которых сохраняется жизнеспособность систем.

Привлечение компьютеров существенно раздвинуло границы моделирования экологических процессов. С одной стороны, появилась возможность всесторонней реализации сложных математических моделей, не допускающих аналитического исследования, с другой - возникли принципиально новые направления, и прежде всего - имитационное моделирование.

4.2. МОДЕЛИ ВНУТРИВИДОВОЙ КОНКУРЕНЦИИ

Рассмотрим простейшую из указанных моделей для вида с дискретными периодами размножения, в которой численность популяции в момент времени t равна N , и изменяется во времени пропорционально величине основной чистой скорости воспроизводства R . Такими видами являются, например, большая часть растений, некоторые виды насекомых, у которых разные поколения четко разнесены во времени. Коэффициент R характеризует количество особей, которое воспроизводится в расчете на одну существующую, а также выживание уже существующих. Данная модель может быть выражена уравнением

$$N_{t+1} = N_t \cdot R, \quad (7.60)$$

решение которого имеет вид

$$N_t = N_0 \cdot R^t, \quad (7.61)$$

где N_0 - начальная численность популяции. Эта модель, однако, описывает популяцию, в которой отсутствует конкуренция и в которой R является константой; если $R > 1$, то численность популяции будет бесконечно увеличиваться. В реальности в какой-то момент начинают работать механизмы сдерживания роста популяции. В литературе приводится немало интересных примеров быстрого роста численности популяций, если бы для их размножения существовали идеальные условия. Особенно это относится к насекомым, растениям и микроорганизмам, которые могли бы покрыть земной шар толстым слоем, если им создать благоприятные условия для размножения. Но в действительности такого роста популяций, когда их численность увеличивается в геометрической прогрессии, на сколько-нибудь длительных промежутках времени не наблюдается.

Следовательно, в первую очередь необходимо изменить уравнение (7.60) таким образом, чтобы чистая скорость воспроизводства зависела от внутривидовой конкуренции.

Конкуренцию можно определить как использование некоего ресурса (пищи, воды, света, пространства) каким-либо организмом, который тем самым уменьшает доступность этого ресурса для других организмов. Если конкурирующие организмы принадлежат к одному виду, то взаимоотношения между ними называют *внутривидовой конкуренцией*, если же они относятся к разным видам, то их взаимоотношения называют *межвидовой конкуренцией*.

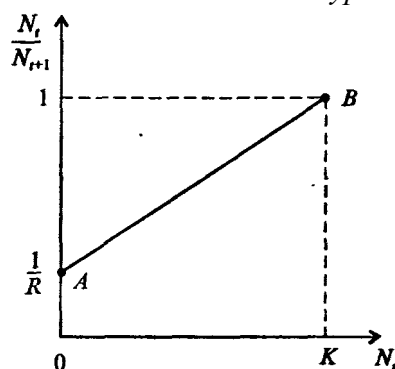


Рис. 7.37. К вопросу об ограничении скорости роста популяции

На рис. 7.37 показана простейшая возможная зависимость скорости воспроизводства от численности популяции. Точка A отражает ситуацию, в которой численность популяции близка к нулю, конкуренция при этом практически отсутствует, и фактическую скорость воспроизводства вполне можно описывать параметром R в его первоначальном виде. Следовательно, при низкой плотности популяции уравнение (7.60) вполне справедливо. В преобразованном виде оно выглядит так:

$$\frac{N_t}{N_{t+1}} = \frac{1}{R}.$$

Точка B , напротив, отражает ситуацию, в которой численность популяции высока, и в значительной степени проявляется внутривидовая конкуренция. Фактическая скорость воспроизводства в результате конкуренции настолько снижена, что популяция в целом может не более чем восстанавливать в каждом поколении свою численность, потому что количество родившихся особей уравнивается количеством погибших. Гипотезе, отраженной на рис. 7.37, соответствует уравнение

$$N_{t+1} = \frac{N_t \cdot R}{1 + a \cdot N_t}, \quad (7.62)$$

где $a = \frac{R-1}{K}$. Это уравнение представляет собой модель роста популяции, ограниченного внутривидовой конкуренцией. Суть этой модели в том, что константа R в уравнении (7.60) заменена на фактическую скорость воспроизводства, т. е. $\frac{R}{1 + a \cdot N_t}$, которая уменьшается по мере роста численности популяции N_t . Достоинство полученного уравнения заключается в его простоте. Такой тип конкуренции приводит к саморегуляции численности популяции, изображенной на рис. 7.38 (для некоторого набора параметров модели; численное решение).

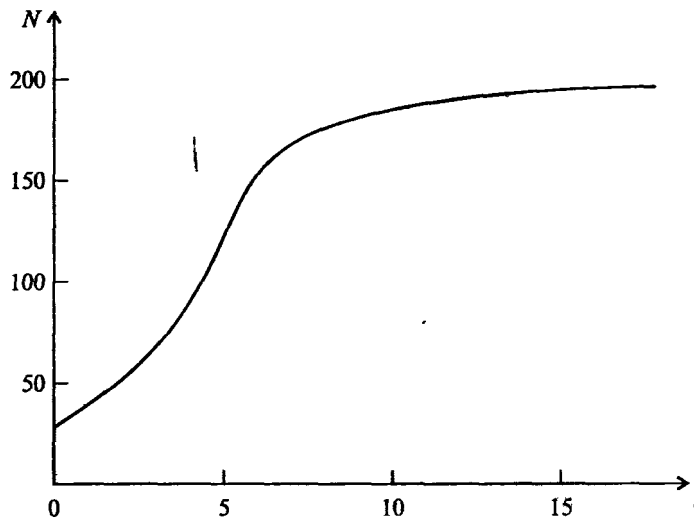


Рис. 7.38. Изменение численности популяции согласно уравнению (7.62) при $R = 2$, $K = 200$, $N_0 = 20$

После несложного изменения в уравнении (7.62) может быть получена гораздо более общая модель, учитывающая интенсивность конкуренции. Простейшая из возможных зависимостей падения скорости роста популяции от ее численности, изображенная на рис. 7.37, является не законом природы, а всего лишь удобной гипотезой. Далеко не всегда реальная динамика численности популяции, определяемая внутривидовой конкуренцией, даже качественно согласуется с изображенной на рис. 7.38. Более общая гипотеза о законе падения скорости роста популяции в зависимости от ее численности приводит к следующему уравнению:

$$N_{t+1} = \frac{N_t \cdot R}{1 + (a \cdot N_t)^b}. \quad (7.63)$$

Общность данной модели в отличие от уравнения (7.62) обусловлена введением в модель параметра b , который определяет тип зависимости падения скорости роста популяции от ее численности.

Набор величин a , b , R можно использовать для сравнения и противопоставления сильно различающихся ситуаций. Другим положительным качеством уравнения (7.63) является его способность освещать новые стороны реального мира. Путем анализа кривых динамики популяций, полученных с помощью уравнения, можно прийти к предварительным выводам относительно динамики природных популяций.

На рис. 7.39, а, б, в и г, построенных с помощью численного моделирования, показаны различные варианты динамики численности популяций, полученные с помощью уравнения (7.63) при разном сочетании параметров b и R .

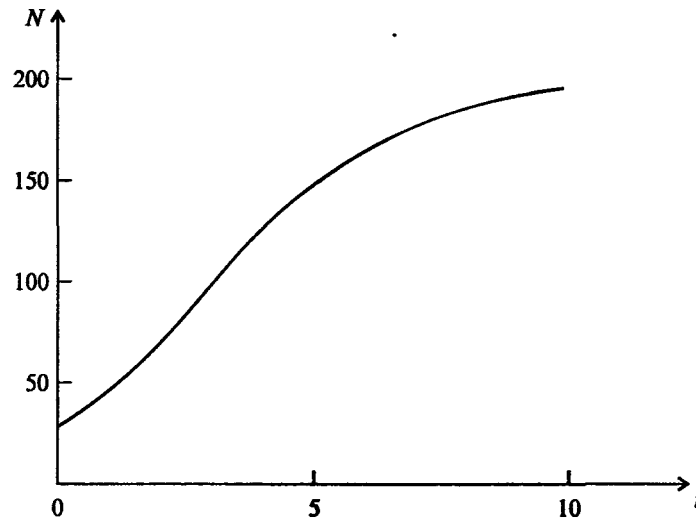


Рис. 7.39, а. Монотонное установление стационарной численности популяции при $b = 1,4$, $R = 2$

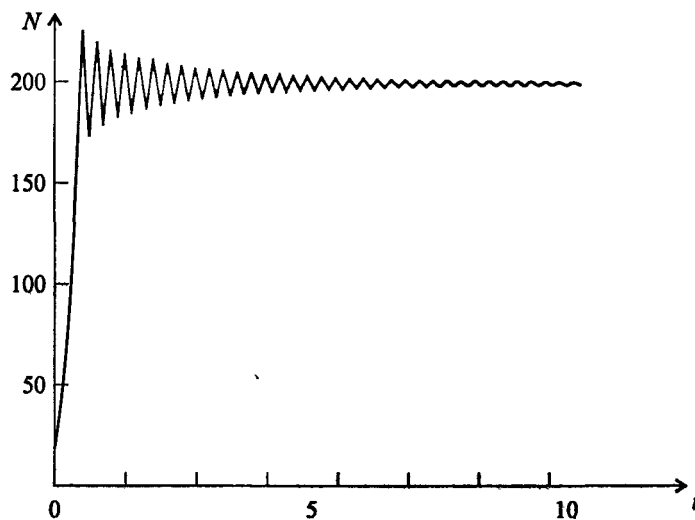


Рис. 7.39, б. Колебательное установление стационарной численности популяции при $b = 3,9$, $R = 2$

Важной частью исследования, связанного с моделью (7.63), является построение на фазовой плоскости (b, R) границ, которые разделяют монотонное затухание, затухающие колебания, устойчивые предельные циклы и случайные (хаотические) изменения, рис. 7.40 Для этого надо задаться значениями a и N_0 и производить расчеты, изменяя параметры b , R . Различить каждый из возможных режимов можно попытаться визуально, выполняя построение на экране компьютера графиков изменения численности популяции и запоминая соответствующие значения параметров b , R при переходе от одного режима к другому. Следует, однако, понимать,

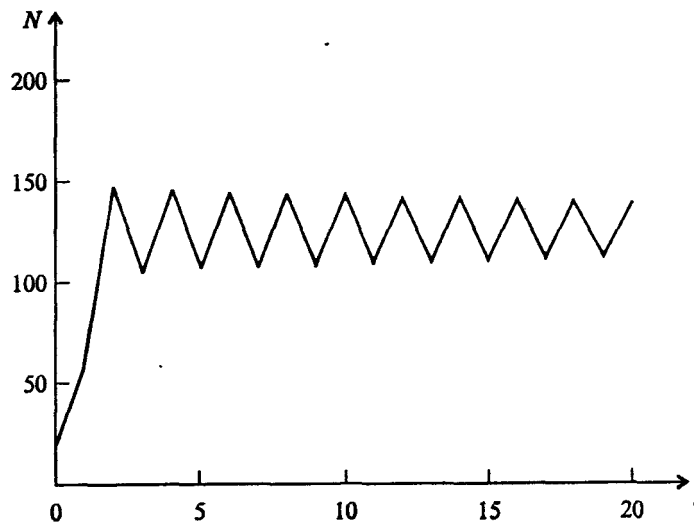


Рис. 7.39, в. Устойчивые предельные циклы изменения численности популяции при $b = 3,6$, $R = 4$

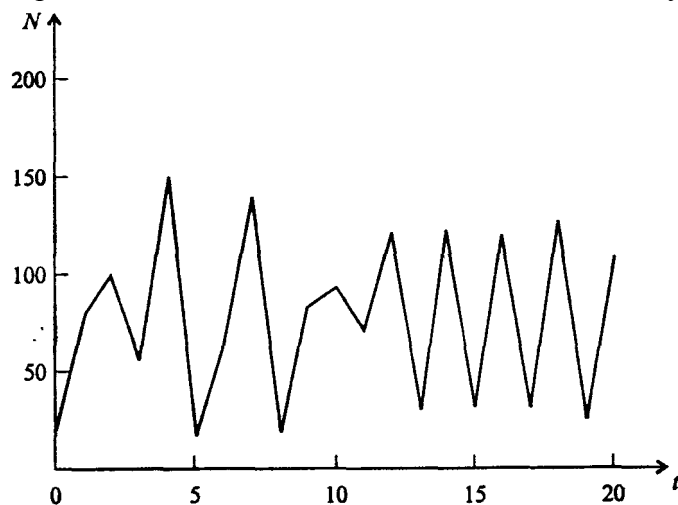


Рис. 7.39, г. Случайные изменения численности популяции при $b = 4,4$, $R = 4$

что установление различия между квазипериодическими, апериодическими и хаотическими движениями - сложная математическая задача. Для ее решения используются методы преобразований Фурье и другие, рассмотрение которых выходит за пределы данной книги.

В связи с уравнениями (7.62), (7.63) сделаем замечание общего характера, справедливое для всех моделей в экологии. В отличие от физики, где «модель» часто является точно установленным законом природы, в экологии модель гораздо более условна. Ее адекватность реальности (в той мере, в какой моделирование вообще на это претендует) проверяется лишь экспериментально. Поскольку экологи много раз наблюдали каждую из динамик, изображенных на рис. 7.39, можно сделать вывод о полезности соответствующей модели (иначе она была бы лишь математическим упражнением).

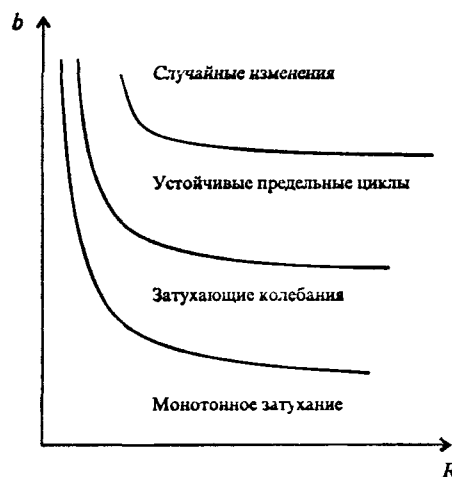


Рис. 7.40. Схематическое изображение фазовой диаграммы динамики численности популяции с дискретным размножением

До сих пор мы рассматривали модели, применимые к популяциям с дискретными периодами размножения. Но в природе встречаются и популяции, где рождение и гибель организмов происходит непрерывно; для таких популяций модели, выраженные уравнениями из предыдущего пункта, непригодны. Рассмотрим *популяцию с непрерывным размножением* и построим модель изменения ее численности. Математическим аппаратом здесь являются дифференциальные уравнения. Скорость роста в этом случае можно обозначить $\frac{\partial N}{\partial t}$, тогда средняя скорость увеличения численности в расчете на одну особь определяется величиной $\frac{\partial N}{\partial t} \cdot \frac{1}{N}$.

Без учета внутривидовой конкуренции получаем $\frac{\partial N}{\partial t} \cdot \frac{1}{N} = r$ или

$$\frac{dN}{dt} = rN.$$

Через r обозначена мгновенная удельная скорость роста численности, т.е. приращение численности за единицу времени в пересчете на одну особь. Согласие этой модели, вследствие того, что не учтена внутривидовая конкуренция, численность популяции при $r > 0$ будет неограниченно возрастать, т.е. будет наблюдаться экспоненциальный рост.

Теперь попробуем учесть внутривидовую конкуренцию. Для этого воспользуемся методом, который однажды уже применяли (см. рис. 7.37).

Обратимся к рис. 7.41. Когда численность популяции близка к нулю, скорость роста определяется величиной r , так как конкуренция еще не оказывает влияния на прирост популяции (точка A). Когда же при возрастании N достигается значение K (предельной плотности насыщения), скорость роста популяции снижается до нуля (точка B).

Записав уравнение прямой AB , получим

$$\frac{dN}{dt} = r \cdot N \cdot \left(\frac{K - N}{K} \right). \quad (7.65)$$

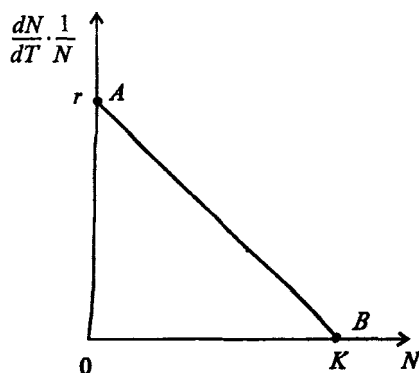


Рис. 7.41. Зависимость скорости воспроизводства от численности популяции

Последнее уравнение известно под названием «логистического». Одно из его достоинств — простота. Уравнение относится к одному из стандартных типов и допускает аналитическое решение:

$$N(t) = \frac{KN_0}{N_0 + (K - N_0)e^{-rt}}. \quad (7.66)$$

В истории экологии логистическое уравнение сыграло значительную роль, оказав большое влияние на применение в экологии математических методов. С другой стороны, простота уравнения накладывает ограничения на область его применения, так как с его помощью можно описать немногие реальные популяции. Но, несмотря на все ограничения, логистическое уравнение ис-

пользуется как составная часть и в других моделях динамики численности популяции, рассматриваемых ниже.

Изменение численности популяции в соответствии с формулами (7.65)-(7.66) иллюстрирует рис. 7.42.

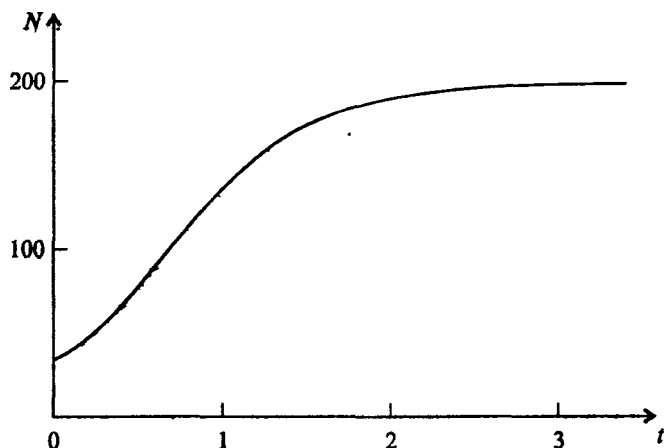


Рис. 7.42. Динамика численности популяции при $N_0 = 25$, $K = 200$, $r = 3$

4.3. ЛОГИСТИЧЕСКАЯ МОДЕЛЬ МЕЖВИДОВОЙ КОНКУРЕНЦИИ

Сущность межвидовой конкуренции заключается в том, что у особей одного вида уменьшается плодовитость, выживаемость и скорость роста в результате использования того же ресурса, что и особями другого вида, причем влиянию конкуренции в той или иной мере подвергаются особи обоих видов.

Так как мы имеем дело с двумя различными популяциями, то введем соответствующие обозначения. Пусть N_1 - численность первой популяции, а N_2 - второй. Предельные плотности насыщения и максимальные врожденные скорости роста популяций обозначим соответственно K_1 , K_2 , r_1 и r_2 , рис. 7.43.

Обратимся к логистическому уравнению (7.65) и попробуем учесть в нем межвидовую конкуренцию. Сделаем это так. Предположим, что M особей вида 2 оказывают такое же воздействие на вид 1, как одна особь вида 1. Константу $\frac{1}{M}$ в этом случае называют *коэффициентом конкуренции*. Обозначим ее α_{12} . Таким образом, чтобы отразить суммарное воздействие на вид 1, надо в логистическом уравнении в числителе дроби вместо N записать $N_1 + \alpha_{12} \cdot N_2$. Аналогично получается уравнение для исследования численности второй популяции. В результате получаем систему двух дифференциальных уравнений:

$$\begin{cases} \frac{dN_1}{dt} = r_1 \cdot N_1 \cdot \frac{K_1 - N_1 - \alpha_{12} \cdot N_2}{K_1}, \\ \frac{dN_2}{dt} = r_2 \cdot N_2 \cdot \frac{K_2 - N_2 - \alpha_{21} \cdot N_1}{K_2}. \end{cases} \quad (7.67)$$

Модель межвидовой конкуренции, выраженная этой системой, названа в честь ее авторов «моделью Лотки-Вольтерры».

Заметим, что если коэффициенты α_{12} или α_{21} больше единицы, то влияние со стороны конкурирующей популяции на особей данного вида сильнее, чем со стороны особей своего вида.

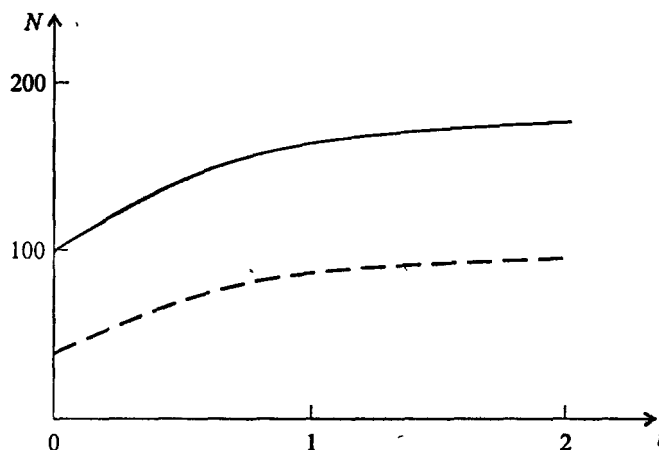


Рис. 7.43. Устойчивое сосуществование популяций при $r_1 = 2$, $r_2 = 4$, $K_1 = 200$, $K_2 = 180$, $\alpha_{12} = 0,5$, $\alpha_{21} = 0,65$, $N_1^{(0)} = 100$, $N_2^{(0)} = 25$. Устойчивое сосуществование достигается лишь при $\alpha_{12} \cdot \alpha_{21} < 1$.
Сплошная линия - численность первой популяции, штриховая - второй

Главный вопрос, который интересует исследователя межвидовой конкуренции - при каких условиях увеличивается или уменьшается численность каждого вида? Для ответа на этот вопрос надо построить диаграммы, где были бы изображены все возможные сочетания численности обоих видов. На таких диаграммах численность одного вида откладывают по горизонтальной оси, а другого - по вертикальной. При одних сочетаниях численности будет отмечаться рост выбранной для наблюдения популяции, при других - уменьшение ее численности. Также для каждого из видов можно провести изоклины - линии, вдоль которых не наблюдается ни увеличения, ни уменьшения численности.

Рассмотрим, как можно построить изоклину для первого вида. По определению, для этой линии $\frac{\partial N}{\partial t} = 0$. Из первого уравнения системы (7.67) получаем

$$r_1 \cdot N_1 \cdot (K_1 - N_1 - \alpha_{12} \cdot N_2) = 0.$$

Это равенство выполняется, если какой-либо из множителей равен нулю. Наибольший интерес представляет ситуация, в которой

$$K_1 - N_1 - \alpha_{12} \cdot N_2 = 0,$$

или

$$N_1 = K_1 - \alpha_{12} \cdot N_2.$$

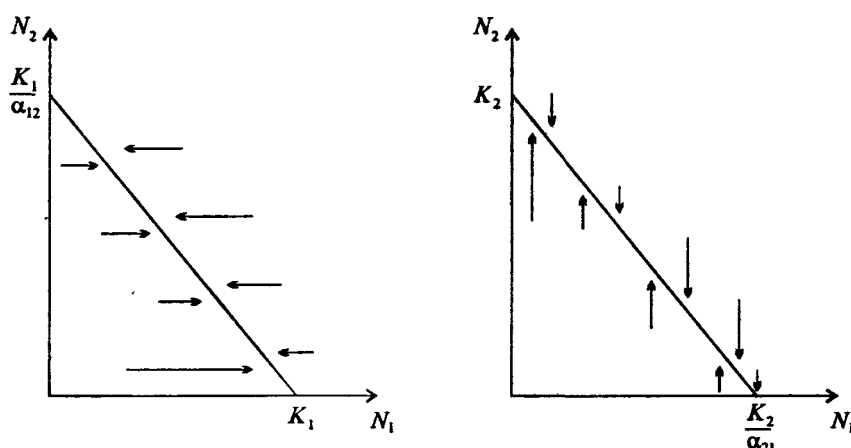


Рис. 7.44. Изоклины, полученные с помощью модели Лотки-Вольтерры. Длины стрелок пропорциональны изменению численности, стрелки указывают направление изменения численности

Таким образом, получено уравнение изоклины, которое, как можно заметить, является уравнением прямой в плоскости (N_1, N_2) . Вверх и вправо от изоклины из-за высокой численности обеих популяций численность вида 1 снижается, в противоположных направлениях - повышается. Аналогично можно построить изоклину для вида 2. На рис. 7.44 построены соответствующие изо-

клины, и показано изменение численности популяций.

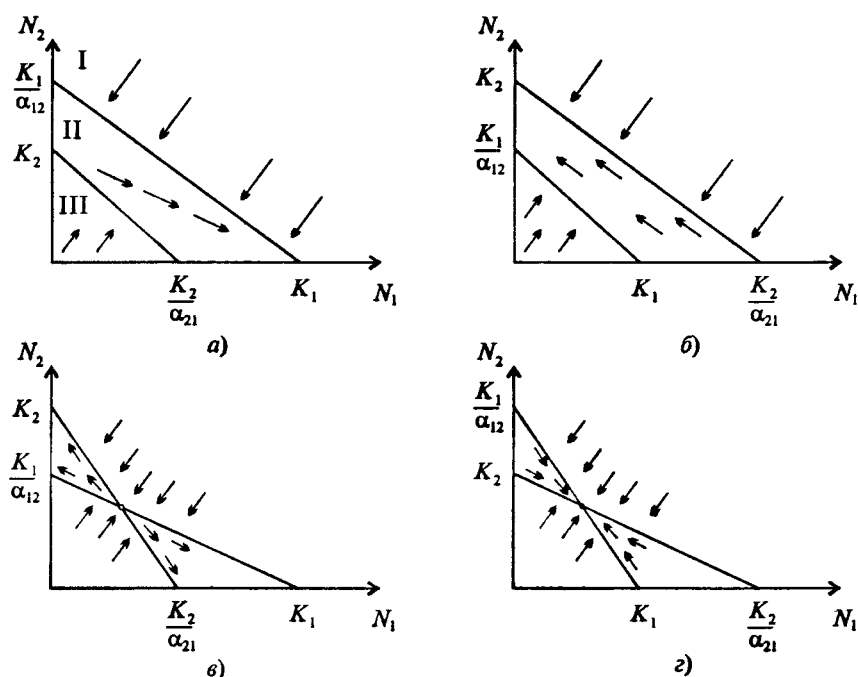


Рис. 7.45. Результаты конкуренции, полученные с помощью модели Лотки-Вольтерры при различных параметрах. На рисунке *а* в зоне I численность обеих популяций падает; в зоне II - численность первой популяции растет, второй - уменьшается; в зоне III - численность обеих популяций увеличивается

Таким образом, получено уравнение изоклины, которое, как можно заметить, является уравнением прямой в плоскости (N_1, N_2) . Вверх и вправо от изоклины из-за высокой численности обеих популяций численность вида 1 снижается, в противоположных направлениях - повышается. Аналогично можно построить изоклину для вида 2. На рис. 7.44 построены соответствующие изоклины и показано изменение численности популяций.

Для решения поставленной выше задачи объединим в одной фазовой плоскости изоклины для обоих видов и будем одновременно исследовать динамику их численности. Изоклины относительно друг друга располагаются четырьмя различными способами, что дает различный исход конкуренции. На рис. 7.45 представлены результаты конкуренции, полученные с помощью системы уравнений (7.67), заимствованные из книги М. Бигона и др. «Экология».

4.4. ДИНАМИКА ЧИСЛЕННОСТИ ПОПУЛЯЦИЙ ХИЩНИКА И ЖЕРТВЫ

Рассматривая динамику численности популяций хищника и жертвы, экологи прежде всего стремятся понять ее закономерности и разъяснить различия между типами динамик. В простейших моделях хищник и жертва рассматриваются безотносительно влияния на них других видов. Одна из самых первых и простых моделей была предложена, как и модель межвидовой конкуренции, Лоткой и Вольтеррой, и носит их имя.

Модель состоит из двух компонентов: C - численность популяции хищника и N - численность популяции жертвы.

Предполагается, что в отсутствие хищника популяция жертвы растет экспоненциально. Чем больше численность той и другой популяции, тем чаще происходят встречи. Число встреченных и съеденных жертв будет зависеть от эффективности, с которой хищник находит и ловит жертву. Если обозначить через a' «эффективность поиска», то скорость поедания жертвы будет равна $a' \cdot C \cdot N$, и окончательно для численности жертвы получаем

$$\frac{dN}{dt} = r \cdot N - a' \cdot C \cdot N.$$

В отсутствие пищи отдельные особи хищника голодают и гибнут. Предположим вновь, что численность хищника в отсутствие пищи будет уменьшаться экспоненциально:

$$\frac{dC}{dt} = -q \cdot C,$$

(q - смертность). Скорость рождения новых особей в данной модели полагается зависящей от двух обстоятельств: скорости потребления пищи $a' \cdot C \cdot N$, и эффективности f , с которой эта пища переходит в потомство хищника. Итак, для численности хищника окончательно получаем

$$\frac{dC}{dt} = f \cdot a' \cdot C \cdot N - q \cdot C.$$

Так как процессы надо рассматривать вместе, объединим уравнения в систему:

$$\begin{cases} \frac{dN}{dt} = r \cdot N - a' \cdot C \cdot N, \\ \frac{dC}{dt} = f \cdot a' \cdot C \cdot N - q \cdot C. \end{cases} \quad (7.68)$$

Как и в предыдущем пункте, свойства этой модели можно исследовать, построив изоклины. Для жертвы имеем

$$\frac{dN}{dt} = 0, \quad r \cdot N - a' \cdot C \cdot N = 0,$$

или, выражая C , получаем

$$C = \frac{r}{a'}.$$

Соответствующее уравнение изоклины для популяции хищника

$$N = \frac{q}{f \cdot a'}.$$

Если поместить обе изоклины на одном рисунке, получим картину взаимодействия популяций (рис. 7.46).

Как видно на рис. 7.47, численности популяций хищника и жертвы совершают периодические колебания: при увеличении численности хищников уменьшается

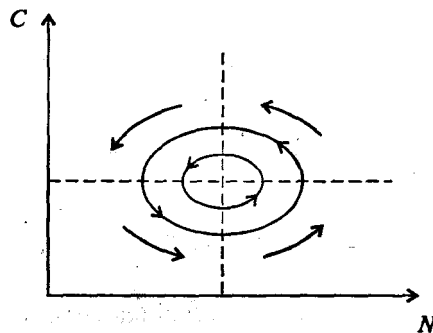


Рис. 7.46. Динамика численности популяции хищника и жертвы. Численность обеих популяций совершает периодические колебания

численность популяции жертвы и наоборот. Такие колебания численности будут продолжаться в соответствии с моделью до тех пор, пока какое-либо внешнее воздействие не изменит численность популяций, после чего произойдет переход в новое устойчивое состояние (такая ситуация называется «нейтральные устойчивые циклы»).

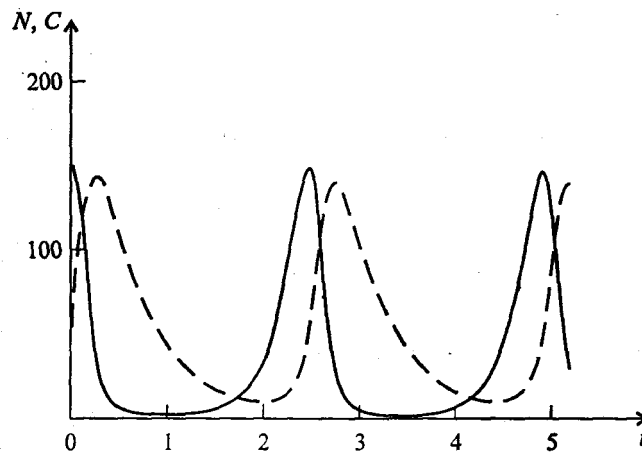


Рис. 7.47. Динамика численности популяции хищника и жертвы при $r = 5$, $a' = 0,1$, $q = 2$, $f = 0,6$, $N_0 = 150$, $C_0 = 50$. Сплошная линия - численность жертвы, штриховая - хищника

4.5. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ ДИНАМИКИ ПОПУЛЯЦИЙ

Выше мы обсудили несколько классических математических моделей в экологии. Известно и немало других, которые можно найти в книгах и в статьях научно-популярных журналов.

Использование математических моделей, основанных на дифференциальных уравнениях или их дискретных аналогах, далеко не единственный путь в компьютерном моделировании динамики популяций. Другую интересную группу составляют чисто стохастические модели или динамические модели с элементами стохастичности. Опишем две возможные модели такого класса: одну - без подробностей, которые рекомендуем продумать самостоятельно, другую - во всех деталях, включая реализующую ее программу.

Первая модель. Две популяции находятся в соотношении «хищник - жертва», описанном выше, но модель совершенно иная. На некотором клетчатом поле (которое наглядно - все или по частям - изображается на дисплее) находятся волки (несколько) и зайцы (много). Зайцы совершают беспорядочные перемещения, длины и направления которых - случайные величины, подчиняющиеся некоторому закону распределения вероятностей, а каждый волк в каждое мгновение ориентируется на ближайшего зайца и может совершать ограниченные прыжки. Если волк в течение некоторого, заранее определенного, промежутка времени остается без добычи (т.е. не попадает в клетку, где находится заяц), то он погибает. Зайцы и волки через определенные моменты размножаются, например, каждый второй заяц приносит двух потомков, каждый второй волк приносит одного потомка, причем скорость размножения волков в два раза ниже, чем зайцев. Все это может красочно изображаться на экране, состояние которого меняется через фиксированные моменты времени.

Это - чисто имитационное моделирование, конечной целью которого является, как и в описанных моделях, установление судьбы популяций в зависимости от многих факторов, входящих в модель. Здесь нет дифференциальных уравнений, но зато в полной мере проявляется стохастика, моделирование случайных процессов. Реализация такой модели требует большого программистского искусства, способствует формированию вероятностных представлений.

Возможны и пограничные - динамико-стохастические модели. Например, в уравнениях Лотки - Вольтерры один из параметров - случайная величина с известным законом распределения. Самостоятельные размышления над такими задачами, даже лишь над их постановкой, очень полезны.

Вторая модель, которую опишем во всех деталях, предложена Д.Конвейем -имитационная модель роста, распада и различных изменений в популяции живых организмов, известная под названием «Жизнь».

	0	...	$j-1$	j	$j+1$	...	n
0							
$\vdots$							
$i-1$			.	.	.		
i			.	Ж	.		
$i+1$			.	.	.		
$\vdots$							
n							

0	1	0	0	0
0	0	1	0	0
0	1	1	1	0
0	0	0	0	0

Рис. 7.48. К имитационной модели «Жизнь»

Для построения алгоритма игры рассмотрим квадратное поле из $n+1$ столбцов и строк с обычной нумерацией от 0 до n , рис. 7.48. Крайние граничные столбцы и строки для удобства определим как «мертвую зону», они играют лишь вспомогательную роль.

Для любой внутренней клетки поля с координатами (i, j) можно определить 8 соседей. Если клетка «живая», ее закрашиваем, если клетка «мертвая», она пустая.

Зададим правила игры. Если клетка (i, j) «живая» и в окружении более трех «живых» клеток, она погибает (от перенаселения). «Живая» клетка также погибает, если в окружении менее двух «живых» клеток (от одиночества). «Мертвая» клетка оживает, если вокруг нее имеются три «живые» клетки. Для удобства введем двумерный массив $A[0..n, 0..n]$, элементы которого принимают значение 0, если соответствующая клетка пустая, и 1, если клетка «живая». Тогда алгоритм определения состояния клетки с координатой (i, j) можно определить следующим образом:

```

S := A[I-1, J-1] + A[I-1, J] + A[I-1, J+1] + A[I+1, J-1]
    + A[I+1, J] + A[I+1, J+1] + A[I, J+1] + A[I, J-1];
If (A[I, J] = 1) And ((S > 3) Or (S < 2)); Then B[I, J] := 0;
If (A[I, J] = 0) And (S = 3) Then B[I, J] := 1;

```

Здесь массив $B[0..n, 0..n]$ определяет координаты поля на следующем этапе. Для всех внутренних клеток от $i = 1$ до $n - 1$ и $j = 1$ до $n - 1$ справедливо сказанное выше. Отметим, что последующие поколения определяются аналогично, лишь стоит осуществить процедуру переприсваивания

```

For I := 1 To N-1 DO For J := 1 To N-1 Do A[I, J] := B[I, J] ;

```

На экране дисплея удобнее выводить состояние поля не в матричном, а в графическом виде. Читатель без труда это может осуществить. Осталось лишь определить процедуру задания начальной конфигурации игрового поля. При случайном определении начального состояния клеток подходит алгоритм

```

For I := 1 To K Do
Begin K1 := Random(N-1); K2 := Random(N-1)+1; A[K1, K2] := 1 End;

```

Интереснее случай, когда начальную конфигурацию мы задаем сами, и удобнее для пользователя делать это непосредственно в графическом виде. Полный текст подобной программы приведен ниже.

Программа 150. «Жизнь».

```

Program PlayLife;
Uses Crt, Graph; Const R = 41;
Var P ; 0..1; A, B : Array[0..R, 0..R] Of 0..1;
    I, J, N, K1, K2, Ss, X, Y, Gm, Gd : Integer; C : Char;
Begin DetectGraph(I, J); InitGraph(I, J, "");
{ -- заставка -- }
For I := 0 To 5 Do
Begin
SetFillStyle (1, I+2); Bar(180+1*10, 100+1*10, 430-1*10, 270-1*10);

```

```

End;
OutTextXY(260, 180, 'ИГРА "ЖИЗНЬ"'); Repeat Until KeyPressed;
{ - начальная конфигурация - }
RestoreCrtMode ;
For I := 0 To R Do
  For J := 0 To R Do
    Begin A[I, J] := 0; B[I, J] := 0 End;
ClrScr;
Write('Начальное поселение будете задавать случайно или вводить (0/1)?');
ReadLn(P);
If P = 0 Then Begin
  Write('Введите начальное количество поселений 2 < n < 200: ');
  ReadLn(N); Gm := VgaHi; SetGraphMode(Gm) ;
  For I := 1 To N Do
    Begin
      K1 := Random(R - 1) + 1; K2 := Random(R - 1) + 1;
      A[K1, K2] := 1; B[K1, K2] := 1;
    End;
  SetLineStyle(0, 0, 1); SetColor(4);
  For I := 0 To R - 1 Do
    Begin
      Line(10 + I * 10, 10, 10 + I * 10, R * 10);
      Line(10, 10 + I * 10, R * 10, 10 + I * 10);
    End;
  For I := 1 To R - 1 Do
    For J := 1 To R - 1 Do
      If A[I, J] = 1 Then
        FloodFill(5 + I * 10, 5 + J * 10, 4);
    Repeat Until KeyPressed;
  End
Else
Begin
  Gm := VgaHi; SetGraphMode(Gm); SetLineStyle(0, 0, 1); SetColor(4);
  For I := 0 To R - 1 Do
    Begin
      Line(10 + I * 10, 10, 10 + I * 10, R * 10);
      Line(10, 10 + I * 10, R * 10, 10 + I * 10);
    End;
  X := 15; Y := 15;
  Repeat
    C := ReadKey; If C = #0 Then C := ReadKey;
    Case C Of
      #72 : Begin
        SetColor(0); Circle(X, Y, 2); Y := Y - 10; SetColor(15);
        Circle(X, Y, 2)
        End;

      #80 : Begin
        SetColor(0); Circle(X, Y, 2); Y := Y + 10; SetColor(15);
        Circle(X, Y, 2)
        End;

      #75 : Begin
        SetColor(0); Circle(X, Y, 2); X := X - 10; SetColor(15);
        Circle(X, Y, 2)
        End;

      #77 : Begin
        SetColor(0); Circle(X, Y, 2); X := X + 10; SetColor(15);
        Circle(X, Y, 2)
        End;
    End;
  Until C = #72;
End;

```

```

#13 : Begin
    FloodFill(X, Y, 4); A[(X-5) Div 10, (Y-5) Div 10] := 1;
    B[(X - 5) Div 10, (Y - 5) Div 10] := 1
End;

#32 : Begin
    SetFillStyled, 0); Bar(X - 4, Y - 4, X + 4, Y + 4);
    A[(X - 5) Div 10, (Y - 5) Div 10] := 0;
    B[(X - 5) Div 10, (Y - 5) Div 10] := 0;
    SetFillStyle(1, 15);
    End;
End {case};
Until C = #27;
SetColor(0); Circle(X, Y, 2) ;
End;
SetColor(15); Repeat Until Key Pressed;
OutTextXY(GetMaxX - 400, GetMaxY - 5, 'для продолжения -любая клавиша, выход
- s');
{- начало игры -- }
Repeat
    For I := 1 To R - 1 Do
    For J := 1 To R - 1 Do
    Begin
        Ss := A[I - 1, J - 1] + A[I - 1, J] + A[I - 1, J + 1];
        Ss := Ss + A[I + 1, J - 1] + A[I + 1, J] + A[I + 1, J + 1] ;
        Ss := Ss + A[I, J - 1] + A[I, J + 1];
        If (A[I, J] = 1) And (Ss > 3) Then
            Begin
                SetFillStyled, 0); Bar(1 + I * 10, 1 + J * 10, 9 + I * 10, 9
                + J * 10);
                B[I, J] := 0; SetFillStyled, 15)
            End;
        If (A[I, J] == 1) And (Ss < 2) Then
            Begin
                SetFillStyled, 0); Bar(1 + I * 10, 1 + J * 10, 9 + I * 10, 9
                + J * 10);
                B[I, J] := 0; SetFillStyled, 15)
            End;
        If (A[I, J] = 0) And (Ss = 3) Then
            Begin
                FloodFill(5 + I * 10, 5 + J * 10, 4); B[I, J] := 1
                End; Delay(20)
            End;
        For I := 1 To R - 1 Do
        For J := 1 To R - 1 Do A[I, J] ••= B[I, J] ;
        C := ReadKey
    Until C = 's'; CloseGraph
End.

```

Теперь приступим к моделированию. Поставим несколько вопросов: какие конфигурации исчезают с течением времени и как быстро? какие структуры могут существовать бесконечно? каковы законы организации структур, их взаимодействия? Перед попыткой ответить на эти вопросы читателю будет полезно поэкспериментировать самому. Предварительно отметим лишь некоторые структуры (рис. 7.49-7.50).



Рис. 7.49. Стационарные структуры, не зависящие от времени

Существуют структуры, которые повторяют себя через 1, 2, ..., n временных шагов, их назы-

вают циклами «-n» (см. рис. 7.50).

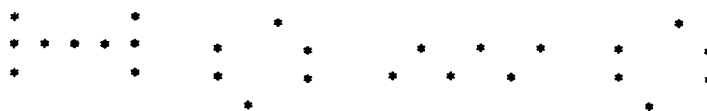


Рис. 7.50. Циклы «-2»

Но наиболее интересны движущиеся структуры - «планеры», «корабли», «паровозы», их столкновения, «аннигиляции» и зарождение новых структур.

Представляет интерес модель игры в одномерном случае. Здесь имеется один ряд клеток. В некоторых из них задаем «жизнь». Требуется изучить эволюцию клеток, если заданы правила их существования и зарождения.

...	0	0	1	1	0	0	1	1	0	1	0	0	...

Определим сумму значений пяти соседних клеток (самой клетки и двух ближайших к ней слева и справа). На следующем шаге в зависимости от суммы, которая может быть равна 0, 1, 2, 3, 4, 5, клеткам присваивается значение 0, 1, 0, 0, 0 или 0, 1, 1, 1, 0, 0.

Пусть $X(N)$ - массив клеток. $X(i) = 0$, если i -я клетка пустая, и $X(i) = 1$, если в i -й клетке имеется «жизнь».

$$S = X(i-2) + X(i-1) + X(i) + X(i+1) + X(i+2).$$

Если $Y(N)$ - массив клеток на следующем временном шаге, то он заполняется, например, по следующему правилу - если $S = 1$, то $Y(i) = 1$, иначе $Y(i) = 0$.

Программа 7.6. «Жизнь на полоске».

```

Program Life1;
Uses Crt, Graph; Const N = 60;
Var J, K : Integer; X, Y : Array(0..N) Of 0..1; I, S, Yg : Integer;
    P : Real; C : Chart;
Begin
  For I := 0 To N Do Begin X[I] := 0; Y[I] := 0 End;
  (начальное поселение)
  X[N Div 2] := 1; DetectGraph(J, K); InitGraph(J, K, ""); Yg := 20;
  For I := 0 To N Do If X[I] = 1 Then Circled * 10 + 10, Yg, 2);
  Repeat C := ReadKey; Yg := Yg + 10;
    For I := 2 To N - 2 Do
      Begin
        S := X[I - 2] + X[I - 1] + X[I] + X[I + 1] + X[I + 2];
        If S In [1..3] Then Y[I] := 1
      End;
  For I := 0 To N Do
    Begin
      If Y[I] = 1 Then Circle(I * 10 + 10, Yg, 2); X[I] := Y[I];
      Y[I] := 0
    End
  Until C = #27; CloseGraph;
End.
```

Контрольные вопросы и задания

1. В чем отличие классической экологии от современной?
2. Какие проблемы рассматриваются в классической экологии?
3. Какие виды взаимодействия организмов принято рассматривать в классической экологии?
4. Какие цели преследуются при составлении математических моделей в экологии?
5. В чем выражается специфика биологических систем в отличие от рассмотренных ранее?

физических и механических систем?

6. Что понимают под конкуренцией в биологии? внутривидовой конкуренцией? межвидовой конкуренцией? Каковы источники конкуренции, и как конкуренция учитывается в приведенных моделях?

7. Какие результаты могут быть получены с помощью простейшей модели роста численности популяции с дискретным размножением? Как изменятся эти результаты, если учесть интенсивность конкуренции?

8. Как построить фазовую диаграмму динамики численности популяции с дискретным размножением?

9. Составьте программу, позволяющую получить все четыре способа изменения численности популяции в модели (7.63).

10. Составьте программу, помогающую построить фазовую диаграмму в модели (7.63).

11. Как выводится логистическое уравнение? Каково аналитическое решение этого уравнения? Как в нем учитывается внутривидовая конкуренция?

12. По какому принципу записывается модель межвидовой конкуренции?

13. Какие результаты могут быть получены с помощью модели межвидовой конкуренции?

14. Какие факторы необходимо учесть при разработке модели системы «хищник -жертва»?

15. Какие результаты могут быть получены с помощью модели «хищник-жертва»?

16. Является ли использование стохастических моделей в исследовании эволюции популяций отражением закономерностей реального мира? В какой мере случайность проявляется в биологических процессах?

17. Получите самостоятельно все результаты, которые приведены в примерах динамики численности популяций с непрерывным размножением и в системе «хищник-жертва», разработав компьютерные программы.

18. Разработайте модель межвидовой конкуренции для трех популяций. Исследуйте ее с помощью компьютерного моделирования.

19. Предусмотрите в модели «хищник-жертва» случайные внешние воздействия, приводящие к изменению численности популяций.

20. Имеется популяция с непрерывным размножением. В течение некоторого времени через равные промежутки часть особей изымается из популяции (собирается «урожай»). При последнем сборе все особи изымаются. Составьте такой план сбора урожая, чтобы суммарный урожай был максимальным. После очередного сбора популяция не должна вымирать. Разработайте соответствующую моделирующую программу.

§5. ГЛОБАЛЬНЫЕ МОДЕЛИ РАЗВИТИЯ ЧЕЛОВЕЧЕСТВА

«Для человеческого ума недоступна совокупность причин явлений. Но потребность отыскивать причины вложена в душу человека. И человеческий ум, не вникнувши в бесчисленность и сложность условий явлений, из которых каждое отдельно может представляться причиной, хватается за первое, самое понятное сближение и говорит: вот причина».

Л. Н. Толстой. Война и мир, т.4, ч.2.

Рассмотренные выше модели представляли собой математические образы отдельных процессов и явлений. Каждая из них интересна по-своему, важна для конкретной науки или вида деятельности. Но все это по своей общечеловеческой важности уступает самому значимому для нас всех вопросу: каково ближайшее будущее человечества как вида в целом? Как будет развиваться мир в обозримом будущем?

Подчеркнем, что речь идет не о политических или экономических прогнозах для какой-то конкретной страны или общества - такие прогнозы, во-первых, в стабильных ситуациях тривиальны, а в нестабильных - мало что значат, а во-вторых, представляют собой интерес, в основном, для жителей этой страны или территории. Речь идет именно о человечестве в целом - какое у него (у нас всех, живущих на Земле) будущее?

Люди в текущей жизни имеют много конкретных проблем и мало склонны к таким общим размышлениям. Жизнь отдельного человека слишком коротка, и еще век-другой назад глобальные

изменения в мире на протяжении жизни одного человека были не очень заметны - даже если он жил в достаточно бурную эпоху. Но в XX веке темпы событий ускорились так, как этого никогда не было в истории человечества, и мы стали все чаще слышать предсказания грядущих глобальных катастроф - гибель природы из-за промышленных загрязнений, неведь откуда взявшиеся «озоновые дыры» в защищающей нас от космических излучений стратосфере, истощение средств воспроизведения кислорода из-за массовой вырубке лесов и т.д. Хотя часть этих страхов нагнетается шарлатанами или заинтересованными в запугивании людей деятелями без каких-либо серьезных обоснований, это не означает отсутствия проблемы - наоборот, она более чем реальна. Даже не столь катастрофические события - например, истощение природных ископаемых - может привести к совершенно радикальным переменам в образе жизни человечества, и при этом, в первую очередь, в странах, которые сегодня являются наиболее промышленно развитыми.

Будущее человечества определяется столь огромным количеством процессов, частично им контролируемых, частично нет, и эти процессы столь взаимосвязаны и имеют столь противоречивые последствия, что лишь математическое моделирование их во всей разумной совокупности, реализуемое на современных компьютерах, может дать качественно верный прогноз. При этом замечательно высказанная великим русским писателем мысль, вынесенная в эпиграф, никогда не должна забываться. Как бы велико ни было неизбежное огрубление реальности при таком моделировании, остаются столько обобщенных факторов «первого ранга», что проследить их неизбежную интерференцию не под силу даже самому могучему уму.

Такие модели, получившие название глобальных (всеохватывающих), начали возникать в 70-х годах нашего века. Наиболее известны модели МИР-1, МИР-2, МИР-3, сформулированные и изученные группой сотрудников Массачусетского технологического института (США) под руководством Д.Х.Медоуз и Д.Форрестера. Работы были начаты по поручению «Римского клуба» - международной неправительственной группы выдающихся государственных деятелей, ученых, бизнесменов. Результаты в свое время произвели в западном мире сенсацию, ибо большинство сценариев возможного развития событий вели к результатам, которые можно назвать концом света (разумеется, с точки зрения человечества). Вместе с тем авторы не раз подчеркивали, что речь идет не о заведомо предопределенном будущем, а о выборе путей развития человечества, среди которых есть и ведущие к стабильности.

Что является причиной возможной нестабильности? - Характерной чертой жизни человечества в эпоху после начала промышленной революции стал быстрый - часто экспоненциально быстрый - рост многих показателей. Период удвоения численности населения Земли составляет примерно 40 лет (наличие такого постоянного периода - характерная черта экспоненциального роста). Биологи и экологи хорошо знают, что экспоненциальное наращивание численности популяции чаще всего кончается катастрофой - истощаются источники, поддерживающие ее существование. С точки зрения существования вида это не трагедия (кроме уникальных случаев, когда данный вид весь сводится к одной популяции). Однако, в наше время человечество израсходовало почти все ресурсы для экстенсивного роста и распространения «вширь». Объем промышленного производства в XX веке также растет практически экспоненциально с годовым темпом прироста в среднем 3,3%. Это приводит к истощению природных ресурсов - полезных ископаемых, чистой воды, чистого воздуха. Содержание в атмосфере одного из устойчивых соединений углерода (диоксида) в результате сжигания органического топлива и истощения лесов возросло с начала века на треть; потенциально это ведет к глобальному потеплению на Земле с самыми катастрофическими последствиями. Чем больше людей, тем больше необходимо продуктов питания, и мировой объем вносимых минеральных удобрений растет экспоненциально с периодом удвоения около 15 лет. Ясно и без всякого моделирования, что подобная жизнь с безудержным ростом всего и вся не может длиться долго — а ныне «долго» сопоставимо со сроком жизни двух-трех поколений.

В то же время для глобальных процессов характерно то, что каждый отдельно взятый процесс нельзя однозначно назвать «хорошим» или «плохим» с точки зрения влияния на судьбу человечества. Увеличение производства удобрений ведет к увеличению производства продуктов питания - «хорошо», но оно же ведет к уменьшению запасов чистой пресной воды, которую портят удобрения, попадающие через почву с дождями в реки и подземные источники, ведет к необходимости увеличения производства энергии для добычи удобрений и связанному с этим химическому и тепловому загрязнению почвы и атмосферы и т.д. - «плохо». Взвесить последствия всего этого на развитие человечества можно лишь при комплексном учете всех факторов разом.

В чем же заключаются возможности избежать катастрофических последствий для развития человечества? Были сформулированы следующие три правила, соблюдение которых необходимо для глобальной устойчивости (прийти к ней необходимо после прекращения нынешних процессов неконтролируемого роста).

1. Для возобновимых ресурсов (лес, вода, рыба и т.д.) темпы потребления не должны превышать темпов естественного восстановления.

2. Для невозобновимых ресурсов (уголь, нефть, руды и т.д.) темпы потребления не должны превышать темпов их замены на возобновимые (развитие солнечной и ветровой энергетики, посадка лесов и т.д.) и темпов развития новых технологий для обеспечения смены ресурсов для того, чтобы после исчезновения, к примеру, нефти был обеспечен приток энергии от нового ресурса.

3. Для загрязняющих веществ предельная интенсивность выбросов не должна превышать темпов, с которыми эти вещества перерабатываются или теряют вредные для окружающей среды свойства.

В настоящее время человечество отнюдь не руководствуется этими правилами (хотя и есть соответствующие попытки - например, соглашения о квотах на рыбную ловлю). Если в прошлые века это не представляло опасности для вида в целом, то в наши дни ситуация изменилась. Достаточно сказать, что если бы при сегодняшней численности населения Земли каждый человек потреблял бы столько энергии и других ресурсов, сколько их сегодня потребляет в среднем гражданин США (при существующих технологиях), это привело бы к немедленной вселенской катастрофе.

Модель **WORLD-3 (МИР-3)**. Модель состоит из пяти секторов:

- стойкие загрязнения;
- невозобновимые ресурсы;
- население;
- сельское хозяйство (производство продуктов питания, плодородие земель, освоение земель);
- экономика (промышленное производство, производство услуг, рабочие места). Исходными являются первичные взаимосвязи, такие как
 - численность населения и запасы промышленного капитала;
 - численность населения и площадь возделываемых земель;
 - площадь возделываемых земель и объем промышленного капитала;
 - численность населения и капитал сектора услуг;
 - капитал сектора услуг и промышленный капитал и т.д.

В каждом секторе прослеживаются все первичные взаимосвязи и выражаются математическими соотношениями. По мере необходимости учитываются процессы материального и информационного запаздывания, так как реакция, скажем, численности населения на улучшение питания является не мгновенной, а запаздывающей. Это типично для большинства рассматриваемых процессов.

Модель WORLD-3 по приведенной в начале главы классификации носит черты дескриптивные и оптимизационные. Ее основное назначение - представить возможные пути достижения экономикой (в широком смысле термина) такой численности населения планеты, которая может поддерживаться окружающей средой неопределенно долгое время. Она не предсказывает нечто отдельное для России или Египта, не решает никаких локальных вопросов. Модель исходит из того, что на Земле существует глобальное сообщество.

Динамика численности населения - интегральная характеристика, которая вбирает в себя все факторы. Чисто умозрительно возможны два типа устойчивых динамик (непрерывный рост и «сигмоидное» приближение к равновесию) и три типа неустойчивых, связанных с выходом за пределы допустимого (колебания с последующим выходом на стационар, хаотические колебания и коллапс, т.е. исчезновение вида). Непрерывный рост представляется совершенно нереалистичским, последняя из неустойчивых динамик - трагедией для человечества, а за резкими колебаниями, как нетрудно догадаться, стоят войны, эпидемии, голод-то, что мы и без всяких моделей видим в реальности.

Типичные для модели WORLD взаимосвязи, находящиеся выражения математическими средствами (дифференциальными и «обычными» уравнениями) приведены на рис. 7.51. Он демонстрирует связи между численностью населения, промышленным капиталом, площадью возделываемых земель.

ваемых земель и загрязнением окружающей среды. Каждая стрелка на рисунке указывает наличие причинной связи, которая может быть непосредственной или запаздывающей, положительной или отрицательной.

Понятия положительной и отрицательной обратной связи взяты из теории автоматического регулирования (раздела кибернетики). Причинно-следственная связь между двумя элементами называется отрицательной, если изменение одного элемента передается второму, возвращается от него к первому и изменяет его в направлении, противоположном первоначальному (подавляет), и положительной, если это изменение, возвращаясь к первому, усиливает его. Если элементов не два, а больше, то говорят о контуре обратной связи, через которую сигнал проходит по кругу, возвращаясь к источнику и влияя на него.

Некоторый набор таких рисунков графически исчерпывает модель WORLD. Однако, за каждой стрелкой - первичные взаимосвязи, за каждой из них - уравнения, в которые входит ряд параметров. Фактически именно значения этих параметров и определяют результаты решения, поэтому к их анализу привлекаются как многочисленные узкие специалисты, так и многие эмпирические (статистические) данные, собранные в десятках справочников, отчетов ООН и отдельных государств. Количество взаимосвязанных переменных в модели WORLD-3 равно 225, параметров - еще больше.



Рис. 7.51. Контурные обратных связей численности населения, капитала, сельскохозяйственного производства и загрязнения окружающей среды

Результаты глобального моделирования. Опубликованные «сценарии» развития человечества, следующие из моделей WORLD, охватывают промежуток времени от 1900 до 2100 гг. Первые 90 лет, уже прошедшие, позволяют «настраивать» модель, понять степень ее достоверности.

Первый из сценариев основан на гипотезе, что все будет развиваться без серьезных изменений, глобальных политических катаклизмов, без особых усилий по сохранению ресурсов и уменьшению загрязнения окружающей среды. Весьма печальные результаты такого развития иллюстрирует рис. 7.52 (заимствовано из того же источника).

Разумеется, временные шкалы здесь весьма расплывчаты. Что будет, если на Земле обнаружатся и окажутся доступными (например, в океанах) дополнительные залежи нефти и газа, других ресурсов? Моделирование безжалостно утверждает, что это не изменит качественно характер эволюции, а лишь сдвинет вправо точки экстремумов кривых.

Вместе с тем модель WORLD позволяет нащупать пути регулируемого развития, которое ведет к плавному («сигмоидному») поведению основных переменных. Этот путь связан с само-

ограничениями и переходом на усовершенствованные промышленные и сельскохозяйственные технологии. Иллюстрацией тому является рис. 7.53.

Итак, математическое компьютерное моделирование вторгается даже в такую сверхзадачу, как самосохранение человека как вида.

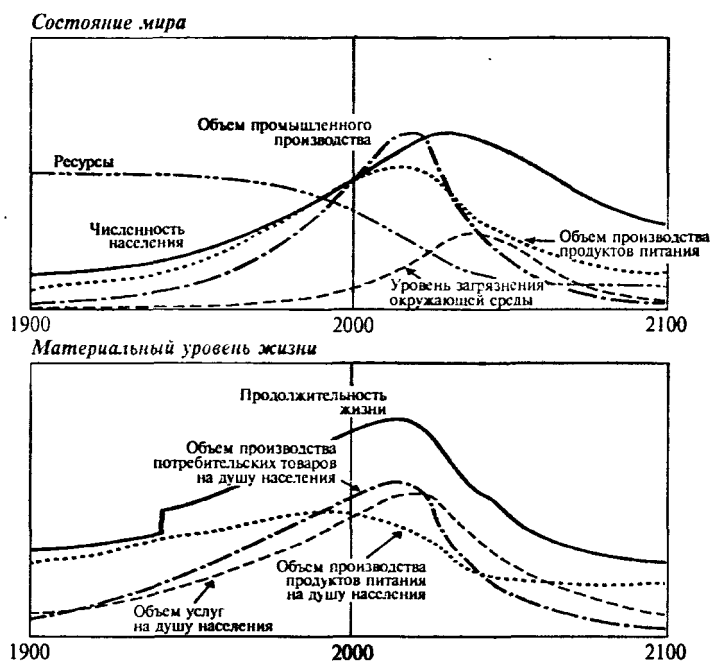


Рис. 7.52. Возможный вариант развития человечества на период 1900-2100 гг.

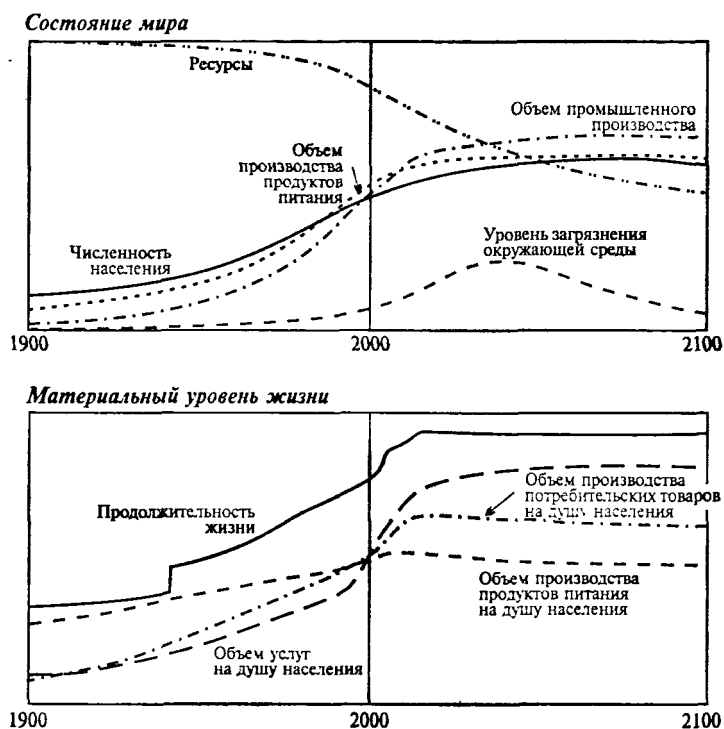


Рис. 7.53. Более оптимистичный вариант развития человечества на тот же период времени

§ 6. МОДЕЛИРОВАНИЕ СЛУЧАЙНЫХ ПРОЦЕССОВ

6.1. ТЕХНИКА СТОХАСТИЧЕСКОГО МОДЕЛИРОВАНИЯ

Понятие «случайный» - одно из самых фундаментальных как в математике, так и в повседневной жизни. Моделирование случайных процессов - мощнейшее направление в современном математическом моделировании.

Событие называется случайным, если оно достоверно непредсказуемо. Случайность окру-

жает наш мир и чаще всего играет отрицательную роль в нашей жизни. Однако есть обстоятельства, в которых случайность может оказаться полезной.

В сложных вычислениях, когда искомый результат зависит от результатов многих факторов, моделей и измерений, можно сократить объем вычислений за счет случайных значений значащих цифр. Из теории эволюции следует, что случайность проявляет себя как конструктивный, позитивный фактор. В частности, естественный отбор реализует как бы метод проб и ошибок, отбирая в процессе развития особи с наиболее целесообразными свойствами организма. Далее случайность проявляется в множественности ее результатов, обеспечивая гибкость реакции популяции на изменения внешней среды.

В силу сказанного имеет смысл положить случайность в основу методов получения решения посредством проб и ошибок, путем случайного поиска.

Отметим, что выше, приведя пример имитационного моделирования - игру «Жизнь», мы уже имели по сути дела стохастическую модель. В данном параграфе обсудим методологию такого моделирования более детально.

Итак, пусть в функционале модели значения некоторых входных параметров определены лишь в вероятностном смысле. В этом случае значительно меняется сам стиль работы с моделью.

При серьезном рассмотрении в обиходе появляются слова «распределение вероятностей», «достоверность», «статистическая выборка», «случайный процесс» и т.д.

При компьютерном математическом моделировании случайных процессов нельзя обойтись без наборов, так называемых, случайных чисел, удовлетворяющих заданному закону распределения. На самом деле эти числа генерирует компьютер по определенному алгоритму, т.е. они не являются вполне случайными хотя бы потому, что при повторном запуске программы с теми же параметрами последовательность повторится; такие числа называют «псевдослучайными».

Рассмотрим вначале генерацию чисел равномерно распределенных на некотором отрезке. Большинство программ - генераторов случайных чисел - выдают последовательность, в которой предыдущее число используется для нахождения последующего. Первое из них - начальное значение. Все генераторы случайных чисел дают последовательности, повторяющиеся после некоторого количества членов, называемого периодом, что связано с конечной длиной машинного слова. Самый простой и наиболее распространенный метод - метод вычетов, или линейный конгруэнтный метод, в котором очередное случайное число x_n определяется «отображением»

$$x_n = (ax_{n-1} + c) \bmod m,$$

где a , c , m - натуральные числа, $\bmod$ - так называемая, функция деления по модулю (остаток от деления одного числа на другое по модулю). Наибольший возможный период датчика (7.69) равен m ; однако, он зависит от a и c . Ясно, что чем больше период, тем лучше; однако реально наибольшее m ограничено разрядной сеткой ЭВМ. В любом случае используемая в конкретной задаче выборка случайных чисел должна быть короче периода, иначе задача будет решена неверно. Заметим, что обычно генераторы выдают отношение $\frac{x_n}{m}$, которое всегда меньше 1, т.е. генерируют последовательность псевдослучайных чисел на отрезке $[0,1]$.

Вопрос о случайности конечной последовательности чисел гораздо сложнее, чем выглядит на первый взгляд. Существует несколько статистических критериев случайности, но все они не дают исчерпывающего ответа. Так, последовательно генерируемые псевдослучайные числа МОГУТ появляться не идеально равномерно, а проявлять тенденцию к образованию групп (т.е. коррелировать). Один из тестов на равномерность состоит в делении отрезка $[0, 1]$ на M равных частей - «корзин», и помещения каждого нового случайного числа в соответствующую «корзину». В итоге получается гистограмма, в которой высота каждого столбика пропорциональна количеству попавших в «корзину» случайных чисел (рис. 7.54).



Рис. 7.54. Вид гистограммы для равномерно распределенных на отрезке $[0,1]$ чисел при достаточно большой выборке

Понятно, что при большом числе испытаний высоты столбиков должны быть почти одинаковыми. Однако, этот критерий является необходимым, но не достаточным; например, он «не замечает» даже очень короткой периодичности. Для не слишком требовательного пользователя обычно достаточны возможности датчика (генератора) случайных чисел, встроенного в большинство языков программирования. Так, в PASCAL есть функция `random`, значения которой - случайные числа из диапазона $[0, 1)$. Ее использованию обычно предшествует использование процедуры `randomize`, служащей для начальной «настройки» датчика, т.е. получения при каждом из обращений к датчику разных последовательностей случайных чисел. Для задач, решение которых требует очень длинных некоррелированных последовательностей, вопрос усложняется и требует нестандартных решений. Равномерно распределенные случайные числа - простейший случай. Располагая датчиком случайных чисел, генерирующим числа $r \in [0, 1]$, легко получить числа из произвольного интервала $[a, b]$.

$$X = a + (b - a) \cdot r.$$

Более сложные распределения часто строятся с помощью распределения равномерного. Упомянем здесь лишь один достаточно универсальный метод Неймана (часто называемый также методом отбора-отказа), в основе которого лежит простое геометрическое соображение. Допустим, что необходимо генерировать случайные числа с некоторой нормированной функцией распределения $f(x)$ на интервале $[a, b]$. Введем положительно определенную функцию сравнения $w(x)$ такую, что $w(x) = \text{const}$ и $w(x) > f(x)$ на $[a, b]$ (обычно $w(x)$ равно максимальному значению $f(x)$ на $[a, b]$). Поскольку площадь под кривой $f(x)$ равна для интервала $[x, x + dx]$ вероятности попадания x в этот интервал, можно следовать процедуре проб и ошибок. Генерируем два случайных числа, определяющих равновероятные координаты в прямоугольнике $ABCD$ с помощью датчика равномерно распределенных случайных чисел:

$$x = a + (b - a) \cdot r, \quad y = w \cdot r$$

и если точка $M(x, y)$ не попадает под кривую $f(x)$, мы ее отбрасываем, а если попадает - оставляем (рис. 7.55). При этом множество координат x оставленных точек оказывается распределенным в соответствии с плотностью вероятности $f(x)$.

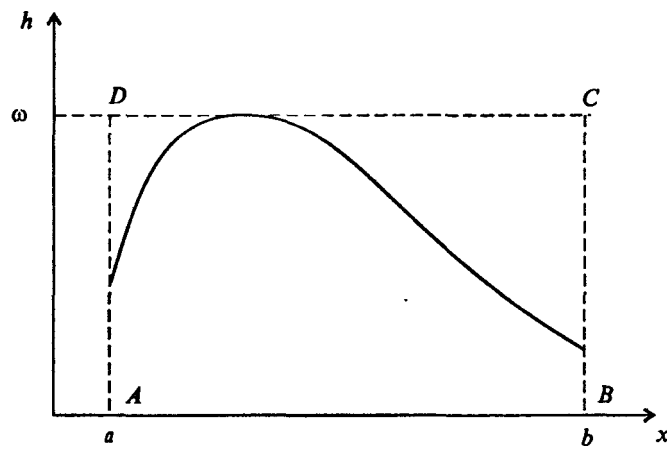


Рис. 7.55. Метод отбора-отказа. Функция $w(x) = f_{\max}$

Этот метод для ряда распределений не самый эффективный, но он универсален, прост и понятен. Эффективен он тогда, когда функция сравнения $w(x)$ близка к $f(x)$. Заметим, что никто не заставляет нас брать $w(x) = \text{const}$ на всем промежутке $[a, b]$. Если $f(x)$ имеет быстро спадающие «крылья», то разумнее взять $w(x)$ в виде ступенчатой функции.

6.2. МОДЕЛИРОВАНИЕ СЛУЧАЙНЫХ ПРОЦЕССОВ В СИСТЕМАХ МАССОВОГО ОБСЛУЖИВАНИЯ

Кому не случалось стоять в очереди и с нетерпением прикидывать, успеет ли он сделать покупку (или заплатить за квартиру, покататься на карусели и т.д.) за некоторое имеющееся в его распоряжении время? Или, пытаясь позвонить по телефону в справочную и натываясь несколько раз на короткие гудки, нервничать и оценивать - дозвонюсь или нет? Из таких «простых» проблем в начале XX века родилась весьма непростая наука - теория массового обслуживания, использующая аппарат теории вероятностей и математической статистики, дифференциальных уравнений и численных методов. Основоположителем ее стал датский ученый А.К.Эрланг, исследовавший проблемы функционирования телефонных станций.

Впоследствии выяснилось, что новая наука имеет многочисленные выходы в экономику, военное дело, организацию производства, биологию и экологию; по ней написаны десятки книг, тысячи журнальных статей.

Компьютерное моделирование при решении задач массового обслуживания, реализуемое в виде метода статистических испытаний (метода Монте-Карло), хоть и не является в теории массового обслуживания основным, но играет в ней важную роль. Основная линия в ней - получение результатов аналитических, т.е. представленных формулами. Однако, возможности аналитических методов весьма ограничены, в то время как метод статистических испытаний универсален и весьма прост для понимания (по крайней мере кажется таковым).

Типичная задача: очередь к одному «продавцу». Рассмотрим одну из простейших задач данного класса. Имеется магазин с одним продавцом, в который случайным образом входят покупатели. Если продавец свободен, он начинает обслуживать покупателя сразу, если покупателей несколько, выстраивается очередь.

Вот аналогичные задачи:

- ремонтная зона в автохозяйстве и автобусы, сошедшие с линии из-за поломки;
- травмопункт и больные, пришедшие на прием по случаю травмы (т.е. без системы предварительной записи);
- телефонная станция с одним входом (или одной телефонисткой) и абоненты, которых при занятом входе ставят в очередь (такая система иногда практикуется);
- сервер локальной сети и персональные машины на рабочем месте, которые шлют сообщение серверу, способному воспринять разом и обработать не более одного сообщения.

Будем для определенности говорить о магазине, покупателях и продавце. Рассмотрим возникающие здесь проблемы, которые заслуживают математического исследования и, как выясняется, весьма серьезного.

Итак, на входе этой задачи случайный процесс прихода покупателей в магазин. Он является «марковским», т.е. промежутки между приходами любой последовательной пары покупателей - независимые случайные события, распределенные по некоторому закону. Реальный характер этого закона может быть установлен лишь путем многочисленных наблюдений; в качестве простейшей модельной функции плотности вероятности можно взять равномерное распределение в диапазоне времени от 0 до некоторого T - максимально возможного промежутка между приходами двух последовательных покупателей. При этом распределении вероятность того, что между приходами двух покупателей пройдет 1 минута, 3 минуты или 8 минут одинакова (если $T > 8$).

Такое распределение, конечно, малореалистично; реально оно имеет при некотором значении $t = \tau$ максимум и быстро спадает при больших t , т.е. имеет вид, изображенный на рис. 7.56.

Можно, конечно, подобрать немало элементарных функций, графики которых похожи на тот, что изображен на рис. 7.56. Например, семейство функций Пуассона, широко используемых в теории массового обслуживания:

$$p_n(t) = \frac{(\lambda t)^n}{n!} \cdot e^{-\lambda t}, \quad (7.70)$$

где λ - некоторая константа, n - произвольное целое. Функции (7.70) имеют максимум при $\tau = \frac{n}{\lambda}$ и

нормированы: $\int_0^{\infty} p_n(t) dt = 1$.

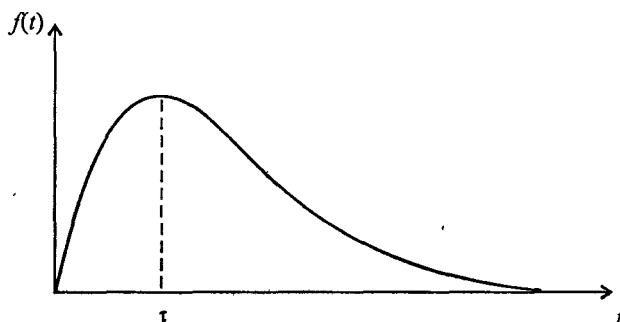


Рис. 7.56. Типичная плотность вероятности распределения времени между приходами покупателей

Второй случайный процесс в этой задаче, никак не связанный с первым, сводится к последовательности случайных событий - длительностей обслуживания каждого из покупателей. Распределение вероятностей длительности обслуживания качественно имеет тот же вид, что и в предыдущем случае; при отработке первичных навыков моделирования методом статистических испытаний вполне уместно принять модель равномерного распределения.

В таблице 7.8 в колонке A записаны случайные числа - промежутки между приходами покупателей (в минутах), в колонке B - случайные числа - длительности обслуживания (в минутах). Для определенности взято $a_{\max} = 10$ и $b_{\max} = 5$. Из этой короткой таблицы, разумеется, невозможно установить, каковы законы распределения приняты для величин A и B ; в данном обсуждении это не играет никакой роли. Остальные колонки предусмотрены для удобства анализа; входящие в них числа находятся путем элементарного расчета. В колонке C представлено условное время прихода покупателя, в колонке D - момент начала обслуживания, E - момент конца обслуживания, F - длительность времени, проведенного покупателем в магазине в целом, G - в очереди в ожидании обслуживания, H - время, проведенное продавцом в ожидании покупателя (магазин пуст). Таблицу удобно заполнять по горизонтали, переходя от строчки к строчке. Приведем для удобства соответствующие формулы (в них $i = 1, 2, 3, \dots$):

$$\begin{aligned} c_1 &= 0, & c_{i+1} &= c_i + a_{i+1}; \\ d_1 &= 0, & d_{i+1} &= \max(c_{i+1}, e_i), \end{aligned}$$

так как начало обслуживания очередного покупателя определяется либо временем его прихода, если магазин пуст, либо временем ухода предыдущего покупателя;

$$\begin{aligned}
e_i &= d_i + b_i; \\
f_i &= e_i - c_i; \\
g_1 &= 0, \quad g_{i+1} = f_{i+1} - b_{i+1}; \\
h_1 &= 0, \quad h_{i+1} = d_{i+1} - e_i.
\end{aligned}$$

Таблица 7.8
Моделирование очереди

N	A	B	C	D	E	F	G	H
1	0	4	0	0	4	4	0	0
2	2	1	2	4	5	3	2	0
3	10	5	12	12	17	5	0	7
4	1	2	13	17	19	6	4	0
5	6	3	19	19	22	3	0	0

Таким образом, при данных случайных наборах чисел в колонках *A* и *B* и покупателям приходилось стоять в очереди (колонка *G*), и продавцу - в ожидании покупателя (колонка *H*).

При моделировании систем такого вида возникают следующие вопросы. Какое среднее время приходится стоять в очереди к прилавку? Чтобы ответить на него, следует найти

$$\bar{g} = \frac{1}{n}(g_1 + g_2 + \dots + g_n)$$

в некоторой серии испытаний. Аналогично можно найти среднее значение величины *h*. Конечно, эти выборочные средние сами по себе - случайные величины; в другой выборке того же объема они будут иметь другие значения (при больших объемах выборки, не слишком отличающиеся друг от друга). Доверительные интервалы, в которых находятся точные средние значения (т.е. математические ожидания соответствующих случайных величин) при заданных доверительных вероятностях находятся методами математической статистики.

Сложнее ответить на вопрос, каково распределение случайных величин *G* и *H* при заданных распределениях случайных величин *A* и *B*. Допустим, в простейшем моделировании мы примем гипотезу о равновероятных распределениях величин *A* и *B* - скажем, для *A* в диапазоне от 0 до 10 минут и *B* - от 0 до 5 минут. Для построения методом статистических испытаний распределений величин *G* и *H* поступим так: найдем в достаточно длинной серии испытаний (реально - в десятках тысяч, что на компьютере делается достаточно быстро) значения $g_{\max}$ (для *H* все делается аналогично) и разделим промежуток $[0, g_{\max}]$ на *m* равных частей - скажем, вначале на 10 - так, чтобы в каждую часть попало много значений g_i . Разделив число попаданий n_k в каждую из частей

на общее число испытаний *n*, получим набор чисел $p_k = \frac{n_k}{n}$ ($k = 1, 2, \dots, n$). Построенные по ним ги-

стограммы дают представление о функциях плотностей вероятности соответствующих распределений. По гистограмме можно составить представление о функции плотности распределения соответствующей случайной величины. Для проверки же гипотезы о принадлежности такого эмпирически найденного распределения тому или иному конкретному виду служат известные статистические критерии.

Располагая функцией распределения (пусть даже эмпирической, но достаточно надежной), можно ответить на любой вопрос о характере процесса ожидания в очереди. Например: какова вероятность прождать дольше *m* минут? Ответ будет получен, если найти отношение площади криволинейной трапеции, ограниченной графиком плотности распределения, прямой $x = m$ и $y = 0$, к площади всей фигуры.

Следующая программа позволяет моделировать описанный выше процесс. На выходе она дает средние значения и дисперсии случайных величин *g* и *h*, полученные по выборке, максимальный объем которой порядка 10000 (ограничение связано с малой допустимой длиной массива в PASCALe; чтобы его смягчить, использовано динамическое описание массивов *g* и *h*). Кроме того, программа строит гистограммы распределений величин *g* и *h*.

Программа 152. Моделирование очереди

```
Program Cohered;
(входной поток равновероятных событий;
динамические массивы позволяют значительно увеличить объем выборки)
Uses Crt, Graph;
Const N = 10000 (число членов выборки);
      W1 = 10 (диапазон времен прихода от 0 до w1});
      W2 = 5 (диапазон времен обслуживания от 0 до w2});
Type T = Array(1..N) Of Real; U = ^T;
Var A, B, C, D, E, F, Aa, Bb, Cc, Dd, Ee, Ff, Dg, Dh, M : Real;
      S1, S2 : Double; I, K, J, I1, I2 : Integer;
      L1, L2, V : Array [1..11] Of Real; G, H : U; Ch : Char;
Begin
  If MaxAvail >= SizeOf(G) Then New(G);
  If MaxAvail >= SizeOf(H) Then New(H);
  Randomize; (ниже - имитационное моделирование)
  Aa := 0; Bb := W2 * Random; Cc := 0; Ee := Bb; Ff := Bb;
  G^[1] = 0; H^[1] := 0;
  For K = 1 To 11 Do
    Begin L1(K) := 0; L2(K) := 0 End;
  For I = 2 To N Do
    Begin
      A := W1 * Random; B := W2 * Random;
      C := Cc + A; If C > Ee Then D := C Else D := Ee;
      E := D + B; F := E - C; G^[I] := F - B; H^[I] := D - Ee;
      Cc := C; Ee := E;
      If G^[I] <= 1 Then L1[1] := L1[1] + 1; If H^[I] = 0 Then
        L2[1] := L2[1] + 1;
      For K := 2 To 10 Do
        Begin
          If (G^[I] > K - 1) And (G^[I] <= K) Then L1[K] := L1[K] + 1;
          If (H^[I] > K - 1) And (H^[I] <= K) Then L2[K] := L2[K] + 1;
        End;
      If G^[I] > 10 Then L1[11] := L1[11] + 1;
      If H^[I] > 10 Then L2[11] := L2[11] + 1;
      S1 := S1 + G^[I]; S2 := S2 + H^[I];
    End;
  For I := 1 To 11 Do (ниже - нормировка распределений g и h)
    Begin
      L1[I] := L1[I] / N; L2[I] := L2[I] / N
    End;
  (ниже - расчет средних и дисперсий величин g и h)
  S1 := S1 / N; S2 := S2 / N; Dg := 0; Dh := 0;
  For I := 1 To N Do
    Begin
      Dg := Dg + Sqr(G^[I] - S1); Dh := Dh + Sqr(H^[I] - S2)
    End;
  Dg := Dg / N; Dh := Dh / N;
  WriteLn('распределение величины g распределение величины h');
  WriteLn;
  For K := 1 To 11 Do
    WriteLn ('11[' , K, ']=', L1[K] : 6 : 4, ' ' : 20, '12(' , K, ']=',
      L2[K] : 6 : 4);
    WriteLn;
  WriteLn('выборочное среднее величины g=', S1 : 6 : 3,
    ' выборочная дисперсия величины g=', Dg : 6 : 3);
  WriteLn('выборочное среднее величины h=', S2 : 6 : 3,
    ' выборочная дисперсия величины h=', Dh : 6 : 3);
  Dispose(G); Dispose(H); WriteLn;
  WriteLn('для продолжения нажать любую клавишу');
```

```

Repeat Until KeyPressed; Ch := ReadKey;
(ниже - построение гистограмм распределений величин g и h)
DetectGraph(I, K); InitGraph(I, K, '');
I := GetMaxX; K := GetMaxY; J := I Div 2; M := L1[1];
For I1 := 2 To 11 Do If L1[I1] > M Then M := L1[I1];
For I1 := 1 To 11 Do V[I1] := L1[I1] / M;
Line(10, K - 10, J - 20, K - 10); Line(10, K - 10, 10, 5);
OutTextXY(20, 100, 'распределение величины g');
For I1 := 1 To 11 Do
  Begin
    I2 := Round((K - 20) * (1 - V[I1])) + 10;
    Line(I1 * 20 - 10, I2, I1 * 20 + 10, I2);
    Line(I1 * 20 - 10, I2, I1 * 20 - 10, K - 10);
    Line(I1 * 20 + 10, I2, I1 * 20 + 10, K - 10);
  End;
Line(J + 20, K - 10, I - 10, K - 10);
Line(J + 20, K - 10, J + 20, 5);
OutTextXY(J + 30, 100, 'распределение величины h'); M := L2[1];
For I1 := 2 To 11 Do If L2[I1] > M Then M := L2[I1];
For I1 := 1 To 11 Do V[I1] := L2[I1] / M;
For I1 := 1 To 11 Do
  Begin
    I2 := Round((K - 20) * (1 - V[I1])) + 10;
    Line(J + I1 * 20, I2, J + I1 * 20 + 20, I2);
    Line(J + I1 * 20, I2, J + I1 * 20, K - 10);
    Line(J + I1 * 20 + 20, I2, J + I1 * 20 + 20, K - 10);
  End;
OutTextXY(200, GetMaxY - 10, 'для выхода нажать любую клавишу');
Repeat Until KeyPressed; CloseGraph
End.

```

Приведем для сравнения результаты расчета средних значений величин g , h и соответствующих среднеквадратичных отклонений Sg , Sh , полученные при одинаковых значениях всех параметров в пяти разных сериях испытания по 10000 событий в серии (табл. 7.9) (входной поток покупателей - процесс равновероятных событий с максимальным временем между приходами 10 мин, длительность обслуживания также распределена равновероятным образом в интервале от 0 до 5 мин).

Таблица 7.9
Сравнение результатов моделирования в разных сериях испытаний

Испытание	g	Sg	h	Sh
1	0,738	1,568	2,508	2,588
2	0,746	1,511	2,500	2,571
3	0,765	1,529	2,446	2,582
4	0,753	1,524	2,451	2,589
5	0,765	1,573	2,482	2,572

Количество цифр выписано таким образом, чтобы отразить значимую разницу между данными разных серий.

Оценим доверительный интервал математических ожиданий величин g и h при с достоверности 0,99 по формуле $\bar{x} - \varepsilon < mx < \bar{x} + \varepsilon$; $\varepsilon = 2,58 \cdot \frac{S}{\sqrt{n}}$ ($\bar{x}$ - среднее значение x ; n - объем выборки; S - среднеквадратичное отклонение). По первой выборке получаем

$$0,738 - 0,025 < mg < 0,738 + 0,025 \text{ (округлим: } 0,71 < mg < 0,77)$$

$$2,508 - 0,067 < mh < 2,508 + 0,067 \text{ (округлим: } 2,44 < mh < 2,58).$$

Таким образом, различия в выборочных средних вполне укладываются в указанные доверительные интервалы.

В рассмотренной задаче, как и в любой более сложной задаче об очередях, может возникнуть критическая ситуация, когда очередь неограниченно растет со временем. В самом деле, если люди заходят в магазин очень часто (или продавец работает слишком медленно), очередь начинает нарастать, и в любой системе с конечным временем обслуживания наступит кризис. Приведем для иллюстрации динамики этого процесса распределения величина - времени ожидания покупателем в очереди и h - времени простоя продавца в ожидании покупателя, при трех наборах параметров $w1$, $w2$, где $w1$ - максимальный интервал времени между приходами покупателей, $w2$ - максимальная длительность обслуживания покупателя (рис. 7.57 - 7.59).

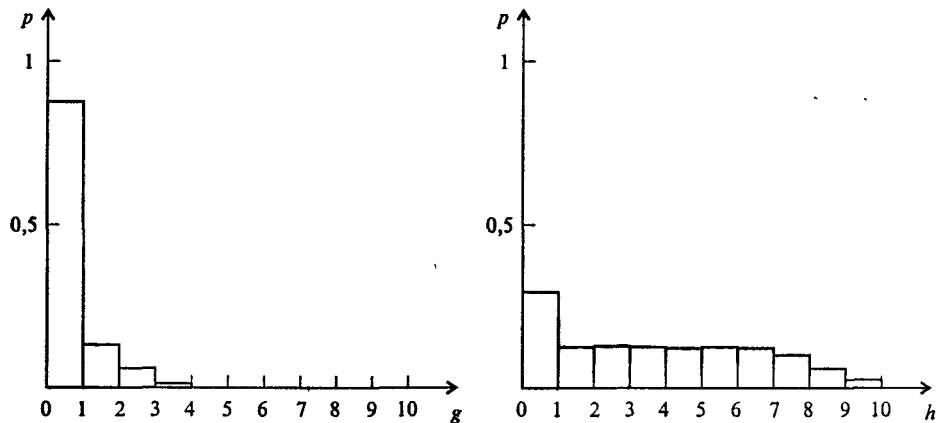


Рис. 7.57. $w1 = 10$, $w2 = 3$ (нет проблем с обслуживанием, вероятность долго простоять в очереди мала, вероятность бездеятельности продавца достаточно велика)

В чем практическое значение задач об очередях? Прежде всего, стремление рационально построить обслуживание потребителей. В магазине можно, к примеру, поставить второго продавца, но если при этом продавцы будут мало заняты, возникает ущерб для предприятия. Таким образом, актуальным является вопрос об отыскании оптимального решения при наличии противоречивых требований, т.е. налицо оптимизационная многокритериальная модель.

Визуально проиллюстрировать формирование очереди поможет следующая программа.

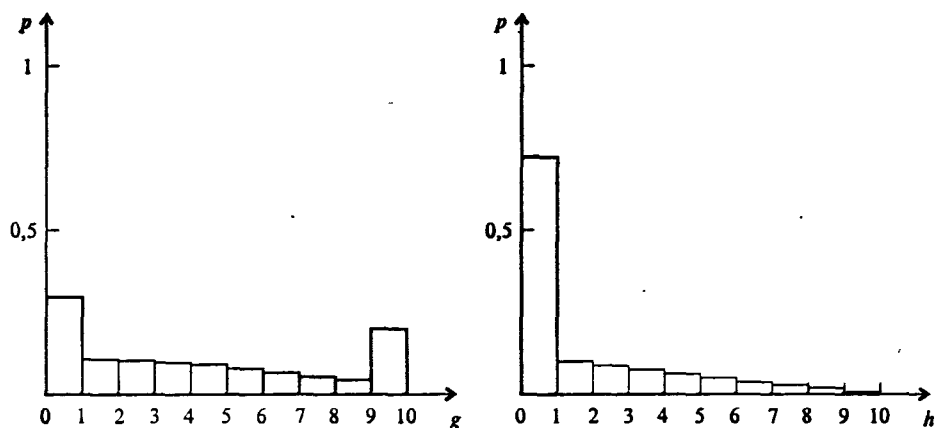


Рис. 7.58. $w1 = 10$, $w2 = 8$ (кризис приближается)

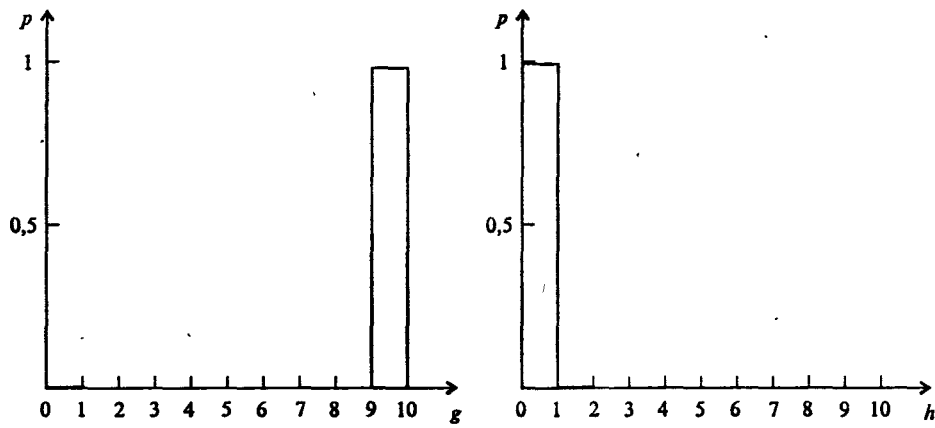


Рис. 7.59. $w1 = 10, w2 = 10$ (кризис наступил)

Программа 153. Имитационное моделирование очереди

```

Program Bank;
Uses Graph, Crt;
Var Gm, Gd, P, X, Qq, I, T, V : Integer; St : String[10];
Begin
  Qq := 0; P := 6; V := 2; Randomize; DetectGraph(Gd, Gm);
  InitGraph(Gd, Gm, ' ');
  SetColor(2); RectAngle(300, 100, 500, 300); T := 0;
  Repeat
    T := T + 1; Str(T, St); SetTextStyle(0, 0, 1) ;
    SetColor(4); OutTextXY(600, 50, St); X := Random(11) ;
    If X < P Then Qq := Qq + 1; SetColor(15) ;
    For I := 0 To Qq Do Circle(490 - I * 30, 200, 15);
    Delay(1000); SetColor(0);
    For I := 0 To Qq Do Circle(490 - I * 30, 200, 15);
    If T Mod V = 0
    Then Begin
      Qq := Qq - 1; If Qq < 0 Then Qq := 0; Setcolor(15) ;
      For I := 0 To Qq Do Circle(490 - I * 30, 200, 15);
    End;
    SetColor(0); OutTextXY(600, 50, St)
  Until KeyPressed Or (Qq > 15); ReadLn;
End.

```

6.3. РАЗЛИЧНЫЕ ПРИМЕРЫ МОДЕЛИРОВАНИЯ СЛУЧАЙНЫХ ПРОЦЕССОВ

Метод статистического моделирования имеет множество приложений. Чаще всего он заключается в том, что для решения математической задачи строится некоторая случайная величина ζ , такая, что математическое ожидание этой случайной величины $E(\zeta)$ является значением искомого решения. Проводя достаточное количество раз эксперимент со случайной величиной ζ , можно найти приближенное решение как среднее значение результатов эксперимента.

1. Вычисление площадей. Найти площадь фигуры G , вписанной в прямоугольник с размерами сторон a и b . С помощью датчика равномерно распределенных случайных чисел многократно генерируются координаты точки, принадлежащей прямоугольнику. Очевидно, что при большом числе испытаний площадь фигуры G приближенно равна отношению числа точек, попавших в область G , к числу всех разыгранных точек. В качестве примера приведем программу вычисления числа π , находя указанным методом площадь круга, вписанного в квадрат, по 100000 испытаний. Оценку точности полученного результата оставляем читателям.

Программа 154. Вычисление числа π методом Монте-Карло

```

Program Chislo_Pi;
Uses Crt; Var I, N : Longint; X, Y : Real;

```

```

Begin
  Randomize; N := 0;
  For I := 1 To 100000 Do
    Begin
      X := Random; Y := Random;
      If Sqr(X - 0.5) + Sqr(Y - 0.5) < 0.25 Then N := N + 1
    End;
  WriteLn ('pi=', (N / 100000 / Sqr(0.5)) : 8 : 5) ;
  Repeat Until KeyPressed End.

```

2. Задача Бюффона. На поле, разграфленное параллельными прямыми, расстояние между которыми L , бросается наугад игла длиной l (рис. 7.60). Какова вероятность того, что игла, упав, пересечет хотя бы одну прямую?

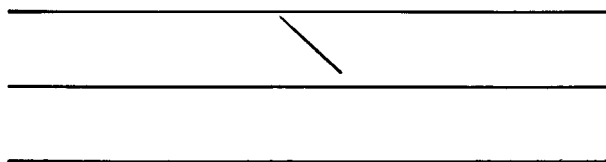


Рис. 7.60. К задаче Бюффона

Ж.Бюффон (XVIII в.) подсчитал: $p = \frac{2l}{\pi L}$. Таким образом, если $L = 2l$, то $p = \frac{1}{\pi}$. Кроме того,

$p = \frac{N_1}{N}$, где N - число бросаний, N_1 - число пересечений иглы с линиями.

Относительная доля случаев, когда игла пересечет хотя бы одну из параллельных прямых равно $p = \frac{1}{\pi}$. Это был один из старинных способов определения числа π .

Имитационное моделирование проведем следующим образом. Примем $L = 1$ и $l = \frac{1}{2}$. «Иглу» будем «бросать» в квадрат размером, скажем, 20×20 , левый нижний угол которого имеет координаты $(0, 0)$. Положение концов иглы будем задавать с помощью датчика равномерно распределенных, случайных чисел в диапазоне от 0 до 20. Точнее говоря, эти числа определяют направление отрезка, вдоль которого находится очередная игла; для того, чтобы ее длина была равна $\frac{1}{2}$, вторую из случайных точек - концов отрезка - подвинем вдоль него до достижения указанной длины иглы. В математическом отношении это сводится к следующей несложной процедуре;

- генерация координат точек $A(x_1, y_1)$, $B(x_2, y_2)$;
- определение координат точки $B_1(x_1 + \alpha(x_2 - x_1), y_1 + \alpha(y_2 - y_1))$, где

$$\alpha = \frac{l}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}.$$

Поскольку расстояние между горизонтальными линиями взято равным единице, а сами линии имеют целочисленные координаты по y , то определить, пересекает ли игла прямую, очень просто - да, если целые части ординат точек A и B_1 различны.

Программа 155 Решение задачи Бюффона.

```

Program Buffon;
  Uses Crt; Var I, J, K, M, N : Integer; X1, X2, Y1, Y2, A1 : Real;
Begin
  Randomize; M := 30000; N := 1;
  For I := 1 To M Do
    Begin
      X1 := Random * 20; Y1 := Random * 20; X2 := Random * 20;
      Y2 := Random * 20;
      A1 := 0.5 / Sqrt(Sqr(X2 - X1) + Sqr(Y2 - Y1)) ;
      J := Round(Y1); K := Round(Y1 + A1 * (Y2 - Y1));

```

```

      If J <> K Then N := H + 1
End;
WriteLn('pi=', W / N) : 8 : 5); Repeat Until KeyPressed
End.

```

Создание демонстрационной программы, которая выводит на экран несколько параллельных прямых из общего набора и имеющие к ним отношение «иглы», предоставляем читателю.

Рекомендуем провести с предложенной программой несколько экспериментов. Понятно, что чем больше значение m , тем, по-видимому, точнее результат. Однако, почему он постоянно слегка занижен? Все ли учтено на краях той зоны, в которой разыгрываются броски иглы? Чтобы почувствовать проблему, следует увеличить число параллельных прямых, что в данной программе очень легко сделать. Почему результат становится лучше? Отметим, что проблема краевых условий, когда события должны по условиям задачи разыгрываться на бесконечном поле, а при имитационном моделировании фактически разыгрываются на конечном (и даже не очень большом), возникает часто и решение ее нетривиально.

3. Нефтяное месторождение. Дано нефтяное месторождение, в котором область залегания нефти G ограничивается кривой C . Дебит скважины, т.е. количество получаемой из нее нефти в единицу времени, зависит от пластового давления нефти U в точке скважины. Поэтому для прогнозирования нефтедобычи важно знать распределение пластового давления на всем месторождении при условии, что оно экспериментально измерено лишь на его границе. В математическом плане функция $U(r)$ удовлетворяет уравнению Лапласа $\nabla^2 U(r) = 0$; задача нахождения его решения внутри области при заданном значении $U(r)$ на границе - так называемая, краевая задача Дирихле; в данной задаче это решение, которое часто совсем не просто найти аналитически, позволило бы правильно определить точку для скважины.

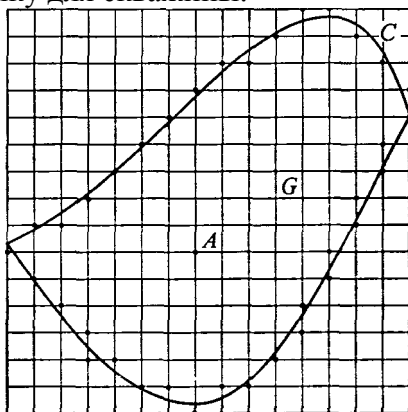


Рис. 7.61. Наложение сетки на заданную область

Покроем область G мелкой сеткой. Отметим узлы, наиболее близкие к границе C , и будем считать, что значения функции U в этих узлах приблизительно равны значениям этой функции в ближайших к ним точках границы. Будем искать значение функции $U(A)$ в некотором внутреннем узле A (рис. 7.61).

Поместим в точке A блуждающую частицу, которая может перемещаться по области в последовательные моменты времени, переходя из одного угла в соседний. Направления перемещений случайны, равновероятны и не зависят от ее положения и предыстории блуждания. Случайный эксперимент состоит в наблюдении факта выхода блуждающей частицы в некоторый граничный узел. Когда блуждание прекращается, запоминается значение функции $f(c_i)$ в этой точке, и так далее, N раз. Замечательный факт состоит в том, что решение в точке

$$U(A) \approx \frac{f(c_1) + f(c_2) + \dots + f(c_N)}{N}.$$

Другими словами, среднее значение приближенно равно решению задачи Дирихле в точке A .

4. Модель «пьяницы» (модель случайного блуждания). Зададим блуждание точки (объекта) по горизонтальной линии по правилу: если случайное число из интервала $[0, 1]$ меньше 0,5, то точка делает шаг вправо $x = x + h$, в противном случае $x = x - h$.

Программа 156. Модель случайного блуждания

```
Program Tochka;
Uses Crt, Graph; Var I, J : Integer; Z, P, X, H, Y : Integer;
Begin
  X := 320; Y := 240; H := 10; P := 4; DetectGraph(I, J) ;
  InitGraphd, J, ");
  SetColor(15); Line(10, 312, 630, 312); Randomize;
  Repeat
    Z := Random(8); If Z >= P Then X := X + H Else X := X - H;
    SetColor(Green); Circle(X, Y, 10); Delay(200);
    SetColor(0); Circle(X, Y, 10)
  Until KeyPressed Or (X >= 640) Or (X <= 0); CloseGraph
End.
```

В программе шаг является постоянным, но никто не мешает нам сделать его переменным, выбирая из интервала $[0, h_{\max}]$ случайным образом. Для этого зададим максимально возможный шаг **HMax** и в цикле определим **H := Random(HMax)**.

Если задать аналогичным образом вероятности движения точки вверх – вниз, вправо - влево ($0 < p_x < 1, 0 < p_y < 1$), получим хаотическое блуждание точки на плоскости. Для моделирования блуждания точки в замкнутом прямоугольном объеме примем абсолютно упругое (зеркальное) отражение от стенок.

Программа 157. Хаотическое блуждание точки

```
Program Broun;
Uses Crt, Graph;
Var I, J, X, Y, HxMax, HyMax, Hx, Hy : Integer; PI, P2, Z1, Z2 : Real;
Begin
  X := 320; Y := 240; HxMax := 30; PI := 0.5; P2 := 0.5; HyMax := 30;
  DetectGraph (I, J) ; InitGraph (1, J, ''); SetColor(15);
  Randomize; RectAngle(100, 100, 540, 380);
  SetColor(Green); Circle(X, Y, 10); Delay(200); SetColor(0);
  Circle(X, Y, 10);
  Repeat
    Z1 := Random; Z2 := Random; Hx := Random(HxMax);
    Hy := Random(HyMax) ;
    If (Z1 < PI) Then X := X + Hx Else X := X - Hx;
    If (Z2 < P2) Then Y := Y + Hy Else Y := Y - Hy;
    If X <= 110 Then X := X + 2 * (110 - X) ;
    If X >= 530 Then X := X - 2 * (-530 + X) ;
    If Y <= 110 Then Y := Y + 2 * (110 - Y) ;
    If Y >= 370 Then Y := Y - 2 * (Y - 370);
    SetColor(Green); Circle(X, Y, 10); Delay(100);
    SetColor(0); Circle(X, Y, 10)
  Until Keypressed; CloseGraph
End.
```

Подобным (хотя и более сложным) образом происходит броуновское движение, хорошо известное из курса физики. Если след точки не стирать, то можно будет наблюдать на экране траекторию такого движения. Нет большого труда перейти к случаю n частиц. Для этого необходимо завести два массива координат точек и аналогично предыдущему примеру организовать их движение.

Программа 158. Броуновское движение

```
Program Gaz;
Uses Crt, Graph;
Var I, J, HxMax, HyMax, Hx, Hy, N, I : Integer;
    X, Y : Array[0..500] Of Integer; PI, P2, Z1, Z2 : Real;
```

```

Begin N := 100;
  For I := 1 To N Do Begin X[I] := 320; Y[I] := 240 End;
  HxMax := 10; PI := 0.5; P2 := 0.5; HyMax := 10;
  DetectGraph (1, J) ; InitGraphd, J, ' '); SetColor(15);
  Randomize; RectAngle(100, 100, 540, 380);
  For I := 1 To N Do PutPixel(X[I], Y[I], White); Delay(200);
  For I := 1 To N Do PutPixel(X[I], Y[I], 0) ;
Repeat
  For I := 1 To N Do
    Begin
      Z1 := Random; Z2 := Random;
      Hx := Random(HxMax); Hy := Random(HyMax);
      If Z1 < PI Then X[I] := X[I] + Hx Else X[I] := X[I] - Hx;
      If Z2 < P2 Then Y[I] := Y[I] + Hy Else Y[I] := Y[I] - Hy;
      If X[I] <= 110 Then X[I] := X[I] + 2 * (110 - X[I]);
      If X[I] >= 530 Then X[I] := X[I] - 2 * (-530 + X[I]);
      If Y[I] <= 110 Then Y[I] := Y[I] + 2 * (110 - Y[I]);
      If Y[I] >= 370 Then Y[I] := Y[I] - 2 * (Y[I] - 370);
      PutPixel (X[I], Y[I], 15)
    End; Delay(100);
  For I := 1 To N Do PutPixel(X[I], Y[I], 0)
Until KeyPressed; CloseGraph
End.

```

Построенная компьютерная модель в первом приближении может позволить моделировать многие явления и процессы, происходящие в газах: рассеивание облака, диффузия газов. С ее помощью можно получить многие зависимости параметров газа друг от друга. В частности, давление (число соударений частиц на стенки) от длины свободного пробега (величин **HxMax** и **HyMax**) или от числа частиц.

Представляет значительный интерес имитационное моделирование явлений в сплошных средах, удовлетворяющих законам идеального газа, таких, как истечение газа в вакуум, ударная волна, волны разрежения и т.п. Для модернизации модели можно ввести в алгоритм упругое столкновение частиц друг с другом, возникновение кластерных ансамблей и многое другое.

При вероятностном моделировании используют различные методы, которые позволяют решать задачи из различных областей. Ниже перечислены сферы применения вероятностных методов.

Метод статистического моделирования: решение краевых задач математической физики, решение систем линейных алгебраических уравнений, обращение матриц и сводящиеся к ним сеточные методы решения систем дифференциальных уравнений, вычисление кратных интегралов, решение интегральных и интегродифференциальных уравнений, задач ядерной физики, газовой динамики, фильтрации, теплотехники.

Метод имитационного моделирования: моделирование систем массового обслуживания, задачи АСУ, АСУП и АСУТП, задачи защиты информации, моделирование сложных игровых ситуаций и динамических систем.

Метод стохастической аппроксимации: рекуррентные алгоритмы решения задач статистического оценивания.

Метод случайного поиска: решение задач оптимизации систем, зависящих от большого числа параметров, нахождение экстремумов функции большого числа переменных.

Другие методы: вероятностные методы распознавания образов, модели адаптации, обучения и самообучения.

Контрольные вопросы и задания

Для ответов на эти вопросы может понадобиться выход за пределы кратких сведений, изложенных в данном параграфе.

1. Какие случайные события называют достоверными? невозможными? несовместимыми? противоположными?

2. Дайте классическое определение вероятности случайного события.
3. В чем заключаются теоремы сложения и умножения вероятностей?
4. Сформулируйте локальную и интегральную теоремы Лапласа для вероятности появления заданного числа случайных событий.
5. Сформулируйте теорему Бернулли для оценки частоты появления случайных событий при независимых повторных испытаниях.
6. Что такое случайная величина дискретная? непрерывная?
7. Дайте определение функции распределения непрерывной случайной величины и плотности распределения.
8. Что такое математическое ожидание и дисперсия случайной величины (при дискретном и при непрерывном распределениях)?
9. Какое распределение называется нормальным? В чем особая значимость нормального распределения в теории вероятностей?
10. Что такое независимая повторная выборка? Как находятся выборочные средние? выборочные дисперсии? В каких связях они с математическим ожиданием и дисперсией случайной величины?
11. Как построить гистограмму выборочного распределения случайной величины? Как по ней судить о функции распределения?
12. Какими свойствами должна обладать точечная оценка параметров функции распределения?
13. Как оценить отклонение выборочного среднего от математического ожидания при малом числе испытаний? при большом числе испытаний? Что такое доверительный интервал?
14. Сформулируйте один из критериев согласия эмпирической и теоретической функций распределения.
15. Что такое «случайное число»? Сформулируйте метод компьютерной генерации последовательности равномерно распределенных псевдослучайных чисел.
16. Сформулируйте один из методов генерации последовательности псевдослучайных чисел с заданным законом распределения.
17. Как формулируются задачи теории массового обслуживания?
18. Какие случайные процессы являются исходными (входными) для обсуждаемой в тексте задачи? Каковы их характеристики?
19. Какие случайные процессы являются объектом исследования (выходными процессами) для обсуждаемой в тексте задачи?
20. Как промоделировать пуассоновский процесс - входной поток клиентов в очередь?
21. Что такое «марковские» случайные процессы и являются ли исследуемые в данном параграфе процессы «марковскими»?
22. С чем связано в первой из приведенных выше программ ограничение на объем выборки? Можно ли его преодолеть и какими способами?
23. Может ли данная программа сделаться несостоятельной при очень большом объеме выборки? Как преодолеть проблему, связанную с периодичностью датчика псевдослучайных чисел?
24. Изучите распределения длительности ожидания в очереди и длительности простоя «продавца» и соответственно средние времена ожидания в системе с одним «прилавком» при различных комбинациях распределений промежутков времен между приходами «покупателей» и времен обслуживания, используя следующие распределения: а) равновероятное; б) пуассоновское; в) нормальное.
25. Выполняя задание 24, возьмите одну из рекомендованных комбинаций параметров и так варьируйте параметр, задающий одно из распределений, чтобы выяснить его критическое значение, переход через которое приводит к неограниченному росту очереди.
26. На междугородной телефонной станции несколько телефонисток обслуживают общую очередь заказов. Очередной заказ обслуживает та телефонистка, которая первой освободилась. Смоделируйте эту ситуацию, обдумайте возникающие проблемы.
27. Пусть на телефонной станции используется обычная система отказа: если абонент занят (и не подключена система «ждите ответа»), очередь не формируется, и необходимо набрать номер вновь. Допустим, что несколько абонентов пытаются связаться с одним и тем же адресатом и в случае успеха разговаривают с ним некоторое (случайное, но не более 3 минут) время. Смодели-

руйте ситуацию. Какова вероятность того, что некто, пытающийся дозвониться, не сможет сделать это за определенное время T ?

28. Одна ткачиха обслуживает несколько ткацких станков, осуществляя по мере неполадок краткосрочное вмешательство, длительность которого - случайная величина. Какова вероятность простоя сразу нескольких станков? Как велико среднее время простоя одного станка? Если задействованы две работницы, что выгоднее: поручить каждой по отдельной группе станков или обеим двоянную группу?

29. Разработайте модель перемешивания (диффузии) газов в замкнутом сосуде и осуществите моделирование с целью изучения закономерностей процесса (зависимости ширины зоны диффузии от числа частиц в газах, их скорости, длины свободного пробега).

30. Разработайте модель поведения газа в плоском канале с поршнем. Рассмотрите случаи вдвигания и выдвигания поршня в замкнутом канале. Изучите поведение ударной волны в зависимости от параметров газа (числа частиц, их скорости, длины свободного пробега)

31. Разработайте модель истечения газа из трубы.

32. Создайте модель «пчелиного роя».

33. Придумайте модель случайного блуждания точки в заданном лабиринте.

34. Предложите модель формирования очереди на стоянке такси.

35. Рассчитайте модель автобусного маршрута с h остановками.

36. Смоделируйте работу продовольственного магазина.

37. Опишите модель автозаправочной станции.

§7. КОМПЬЮТЕРНОЕ МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ В ЭКОНОМИКЕ

7.1. ПОСТАНОВКА ЗАДАЧИ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

В последние годы мы особенно отчетливо ощутили, что нет ничего важнее для общества, чем здоровая экономика. Научное исследование основ функционирования экономики - сложная и интересная деятельность. Математические методы в ней играют возрастающую с каждым десятилетием роль, а реализация возникающих при этом математических моделей и получение практически важных результатов невозможны без ЭВМ.

В данном параграфе рассматривается лишь один из разделов - оптимальное планирование - и внутри него одна из моделей, так называемое, линейное программирование. Это связано с относительной простотой и ясностью как содержательной постановки соответствующих задач, так и методов решения. О таких интересных, но более сложных проблемах, как выпуклое программирование, динамическое программирование, теория игр мы лишь упомянем, отсылая читателей за подробностями к специальной литературе. Отметим еще, что термин «программирование» в названии этих разделов теории оптимального планирования весьма условен, связан с историческими обстоятельствами и к программированию в общепринятом сейчас смысле прямого отношения не имеет.

Общеизвестно, сколь важно для решения экономических задач планирование - как при рыночной, так и при плановой экономике. Обычно для решения экономической проблемы существует много способов (стратегий), отнюдь не равноценных по затратам финансов, людских ресурсов, времени исполнения, а также по достигаемым результатам. Наилучший из способов (по отношению к выбранному критерию - одному или нескольким) называют оптимальным. Приведем простейший пример такого рода задач.

Пример 1. На некотором предприятии могут выпускать изделия двух видов (например, мотоциклы и велосипеды). В силу ограниченности возможностей сборочного цеха в нем могут собирать за день либо 25 мотоциклов (если не собирать вообще велосипеды), либо 100 велосипедов (если не собирать вообще мотоциклы), либо какую-нибудь комбинацию тех и других, определяемую приемлемыми трудозатратами. Склад может принять не более 70 изделий любого вида в сутки. Известно, что мотоцикл стоит в 2 раза дороже велосипеда. Требуется найти такой план выпуска продукции, который обеспечил бы предприятию наибольшую выручку.

Такого рода задачи возникают повседневно в огромном количестве, но в реальности число изделий гораздо больше двух, да и дополнительных условий тоже больше. Решить подобную задачу путем перебора всех мыслимых вариантов часто невозможно даже на ЭВМ. В нашем приме-

ре, однако, в ЭВМ нет необходимости - задача решается очень легко.

Обозначим число выпускаемых за день мотоциклов x , велосипедов - y . Пусть τ_1 - время (в часах), уходящее на производство одного мотоцикла, а τ_2 - одного велосипеда. Из условия задачи следует, что $\tau_1 = 4\tau_2$. Если завод работает круглосуточно, то, очевидно, при одновременном выпуске обоих изделий

$$\tau_1 \cdot x + \tau_2 \cdot y \leq 24,$$

$$4\tau_2 \cdot x + \tau_2 \cdot y \leq 24, \quad 4x + y \leq \frac{24}{\tau_2}.$$

Но $\frac{24}{\tau_2}$ - число максимально производимых велосипедов, равное 100. Итак, возможности производства определяют условие

$$4x + y \leq 100.$$

Еще одно условие - ограниченная емкость склада:

$$x + y \leq 70$$

Обозначим цену мотоцикла a_1 (руб.), цену велосипеда - a_2 (руб.). По условию $a_1 = 2a_2$. Общая цена дневной продукции

$$S = a_1 \cdot x + a_2 \cdot y = 2a_2 \cdot x + a_2 \cdot y = a_2 \cdot (2x + y).$$

Поскольку a_2 - заданная положительная константа, то наибольшего значения следует добиваться от величины $f = 2x + y$.

Итак, учитывая все условия задачи, приходим к ее математической модели: среди неотрицательных целочисленных решений системы линейных неравенств

$$\begin{cases} 4x + y \leq 100, \\ x + y \leq 70 \end{cases} \quad (7.71)$$

найти такое, которое соответствует максимуму линейной функции

$$f = 2x + y. \quad (7.72)$$

Проще всего решить эту задачу чисто геометрически. Построим на плоскости (x, y) область, соответствующую неравенствам (7.71) и условию неотрицательности x и y . Эта область выделена на рис. 7.62 жирной линией. Всякая ее точка удовлетворяет неравенствам (7.71) и неотрицательности переменных. Пунктирные линии на рисунке - семейство прямых, удовлетворяющих уравнению $f = 2x + y = c$ (c разными значениями константы c). Вполне очевидно, что наибольшему возможному значению f , совместному с предыдущими условиями, соответствует жирная пунктирная линия, соприкасающаяся с областью M в точке P .

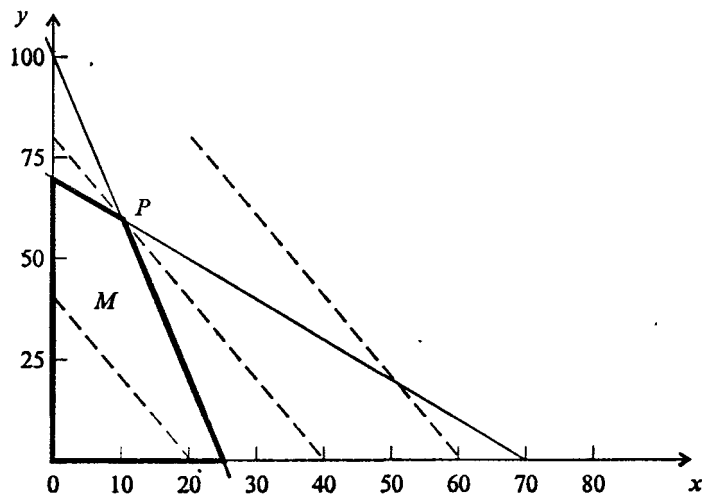


Рис. 7.62. Графическое решение задачи об оптимальном плане производства (к примеру 1)

Этой линии соответствует значение $f = 80$. Пунктирная линия правее хоть и соответствует большему значению f , но не имеет общих точек с M , левее - меньшим значениям f . Координаты точки $P(10, 60)$ - искомый оптимальный план производства.

Отметим, что нам «повезло» - решение (x, y) оказалось целочисленным. Если бы прямые

$$\begin{cases} 4x + y = 100 \\ x + y = 70 \end{cases}$$

пересеклись в точке с нецелочисленными координатами, мы бы столкнулись со значительными проблемами. Еще больше их было бы, если бы наш завод выпускал три и более видов продукции.

Прежде чем обсуждать возникающие при этом математические проблемы, дадим формулировки нескольких классических задач линейного программирования в общем виде.

Пример 2. Транспортная задача. Некий продукт (например, сталь) вырабатывается на m заводах $P_1, P_2, \dots, P_m$, причем ежемесячная выработка составляет $a_1, a_2, \dots, a_m$ тонн, соответственно. Пусть эту сталь надо доставить на предприятия $Q_1, Q_2, \dots, Q_k$ (всего k), причем $b_1, b_2, \dots, b_k$ - ежемесячная потребность этих предприятий. Наконец, пусть задана стоимость c_{ij} перевозки одной тонны стали с завода P_i на предприятие Q_j . Естественно считать, что общее производство стали равно суммарной потребности в ней:

$$a_1 + a_2 + \dots + a_m = b_1 + b_2 + \dots + b_k \quad (7.73)$$

Необходимо составить план перевозок, при котором

- 1) была бы точно удовлетворена потребность в стали предприятий $Q_1, Q_2, \dots, Q_k$,
- 2) была бы вывезена вся сталь с заводов $P_1, P_2, \dots, P_m$;
- 3) общая стоимость перевозок была бы наименьшей.

Обозначим через x_{ij} количество стали (в тоннах), предназначенной к отправке с завода P_i на предприятие Q_j . План перевозок состоит из $(m \cdot k)$ неотрицательных чисел x_{ij} ($i = 1, 2, \dots, m; j = 1, 2, \dots, k$).

Таблица 7.10
Схема перевозок стали

	в Q_1	в Q_2	в Q_3	...	в Q_k	Отправлено
из P_1	x_{11}	x_{12}	x_{13}	...	x_{1k}	a_1
из P_2	x_{21}	x_{22}	x_{23}	...	x_{2k}	a_2
...	...	...	...	...	...	...
из P_m	x_{m1}	x_{m2}	x_{m3}	...	x_{mk}	a_m
Привезено	b_1	b_2	b_3	...	b_k	

Первое условие примет вид

$$\begin{cases} x_{11} + x_{21} + \dots + x_{m1} = b_1, \\ x_{12} + x_{22} + \dots + x_{m2} = b_2, \\ \dots \\ x_{1k} + x_{2k} + \dots + x_{mk} = b_k. \end{cases} \quad (7.74)$$

Второе условие примет вид

$$\begin{cases} x_{11} + x_{12} + \dots + x_{1k} = a_1, \\ x_{21} + x_{22} + \dots + x_{2k} = a_2, \\ \dots \\ x_{m1} + x_{m2} + \dots + x_{mk} = a_m. \end{cases} \quad (7.75)$$

Раз стоимость перевозки одной тонны из P_i в Q_j равна c_{ij} , то общая стоимость S всех перевозок равна

$$S = c_{11}x_{11} + c_{12}x_{12} + \dots + c_{1k}x_{1k} + c_{21}x_{21} + c_{22}x_{22} + \dots + c_{2k}x_{2k} + \dots + c_{m1}x_{m1} + c_{m2}x_{m2} + \dots + c_{mk}x_{mk}. \quad (7.76)$$

Таким образом, мы приходим к следующей чисто математической задаче: дана система $m+k$ линейных алгебраических уравнений (7.74) и (7.75) с $m \cdot k$ неизвестными (обычно $m \cdot k \gg m+k$) и линейная функция S . Требуется среди всех неотрицательных решений данной системы найти такое, при котором функция S достигает наименьшего значения (минимизируется).

Практическое значение этой задачи огромно, ее умелое решение в масштабах нашей страны могло бы экономить ежегодно огромные средства.

Пример 3. Задача о диете. Пусть у врача-диетолога имеется n различных продуктов $F_1, F_2, \dots, F_n$, из которых надо составить диету с учетом их питательности. Пусть для нормального питания человеку необходимо m веществ $N_1, N_2, \dots, N_m$. Предположим, что за месяц каждому человеку необходимо γ_1 кг вещества N_1 , γ_2 кг вещества N_2 , ..., γ_m кг вещества N_m . Для составления диеты необходимо знать содержание питательных веществ в каждом продукте. Обозначим через α_{ij} количество i -го питательного вещества, содержащегося в одном килограмме j -го продукта. Всю эту информацию представляют в виде, так называемой, матрицы питательности (табл. 7.11).

Таблица 7.11
Матрица питательности

Питательное вещество	Продукт			
	F_1	F_2	...	F_n
N_1	α_{11}	α_{12}	...	α_{1n}
N_2	α_{21}	α_{22}	...	α_{2n}
...	...	...	...	...
N_m	α_{m1}	α_{m2}	...	α_{mn}

Предположим, что диетолог уже выбрал диету, т.е. определил, что человек должен за месяц потреблять η_1 кг продукта $F_1, \dots, \eta_n$ кг продукта F_n . Полное количество питательного вещества N_1 будет

$$\eta_1 \alpha_{11} + \eta_2 \alpha_{12} + \dots + \eta_n \alpha_{1n}.$$

По условию требуется, чтобы его, по крайней мере, хватило

$$\eta_1 \alpha_{11} + \eta_2 \alpha_{12} + \dots + \eta_n \alpha_{1n} \geq \gamma_1$$

Точно то же и для остальных веществ. В целом

$$\eta_1 \alpha_{i1} + \eta_2 \alpha_{i2} + \dots + \eta_n \alpha_{in} \geq \gamma_i \quad (i = 1, 2, \dots, m)$$

Эти условия определяют наличие минимума необходимых питательных веществ. Диета, для которой выполнены условия (7.78) - допустимая диета. Предположим, что из всех допустимых диет должна быть выбрана самая дешевая. Пусть π_i - цена 1 кг продукта F_i . Полная стоимость диеты, очевидно,

$$S = \pi_1 \eta_1 + \pi_2 \eta_2 + \dots + \pi_n \eta_n. \quad (7.79)$$

Таким образом, мы пришли к задаче: найти неотрицательное решение $\eta_1, \dots, \eta_n$ системы неравенств (7.78), минимизирующее выражение (7.79).

В примерах, приведенных выше, имеется нечто общее. Каждый из них требует нахождения наиболее выгодного варианта в определенной экономической ситуации. С чисто математической стороны в каждой задаче требуется найти значение нескольких неизвестных так, чтобы

- 1) все эти значения были неотрицательны;
- 2) удовлетворяли системе линейных уравнений или линейных неравенств;
- 3) при этих значениях некоторая линейная функция имела бы минимум (или максимум).

Таким образом, *линейное программирование* - это математическая дисциплина, изучающая методы нахождения экстремального значения линейной функции нескольких переменных при условии, что последние удовлетворяют конечному числу линейных уравнений и неравенств. Запишем это с помощью формул: дана система линейных уравнений и неравенств.

Запишем это с помощью формул: дана система линейных уравнений и неравенств

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2, \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m, \\ a_{m+1,1}x_1 + a_{m+1,2}x_2 + \dots + a_{m+1,n}x_n \geq b_{m+1}, \\ a_{m+2,1}x_1 + a_{m+2,2}x_2 + \dots + a_{m+2,n}x_n \geq b_{m+2}, \\ \dots \\ a_{m+k,1}x_1 + a_{m+k,2}x_2 + \dots + a_{m+k,n}x_n \geq b_{m+k}, \end{cases} \quad (7.80)$$

и линейная функция

$$f = c_1x_1 + c_2x_2 + \dots + c_nx_n \quad (7.81)$$

Требуется найти такое неотрицательное решение

$$x_1 \geq 0, x_2 \geq 0, \dots, x_n \geq 0$$

системы (7.80), чтобы функция f принимала наименьшее (или наибольшее) значение.

Условия (7.80) называют ограничениями данной задачи, а функцию f - целевой функцией (или линейной формой). В приведенных выше примерах ограничения имели вид не уравнений, а неравенств. Заметим, что ограничения в виде неравенств всегда можно свести к системе в виде равенств (способом введения добавочных неизвестных).

Так, для неравенства

$$a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n \geq b_i$$

вводя добавочное неизвестное x_{n+1} , получаем

$$x_{n+1} = a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n - b_i$$

Потребовав его неотрицательности наряду с остальными неизвестными, получим, что условие $x_{n+1} \geq 0$ превращает (7.84) в (7.83). Введя по отдельному дополнительному неизвестному для каждого из неравенств, получим систему уравнений, равносильную исходной системе неравенств.

Пример. Дана система неравенств

$$\begin{cases} 2x + 3y - z \geq 6, \\ x - y + z \geq 2. \end{cases}$$

Сведем ее к системе уравнений. Получим

$$\begin{cases} u = 2x + 3y - z - 6, \\ v = x - y + z - 2. \end{cases}$$

После оптимизации значениями дополнительных неизвестных следует пренебречь.

7.2. СИМПЛЕКС-МЕТОД

Для решения ряда задач линейного программирования существуют специальные методы. Есть, однако, общий метод решения всех таких задач. Он носит название симплекс-метода и состоит из алгоритма отыскания какого-нибудь произвольного допустимого решения и алгоритма последовательного перехода от этого решения к новому допустимому решению, для которого функция f изменяется в нужном направлении (для получения оптимального решения).

Пусть система ограничений состоит лишь из уравнений

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2, \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$

и требуется отыскать минимум линейной функции (7.81). Для отыскания произвольного опорного решения приведем (7.85) к виду, в котором некоторые r неизвестных выражены через остальные, а свободные члены неотрицательны (как это сделать - обсудим позднее):

$$\begin{cases} x_1 = a'_{1,r+1}x_{r+1} + a'_{1,r+2}x_{r+2} + \dots + a'_{1n}x_n + b'_1, \\ x_2 = a'_{2,r+1}x_{r+1} + a'_{2,r+2}x_{r+2} + \dots + a'_{2n}x_n + b'_2, \\ \dots \\ x_r = a'_{r,r+1}x_{r+1} + a'_{r,r+2}x_{r+2} + \dots + a'_{rn}x_n + b'_r. \end{cases}$$

Неизвестные $x_1, x_2, \dots, x_r$ - *базисные неизвестные*, набор $\{x_1, x_2, \dots, x_r\}$ называется базисом, а остальные неизвестные $\{x_{r+1}, x_{r+2}, \dots, x_n\}$ - *свободные*. Подставляя (7.86) в (7.81), выразим функцию f через свободные неизвестные:

$$f = c_0 + c'_{r+1}x_{r+1} + c'_{r+2}x_{r+2} + \dots + c'_n x_n.$$

Положим все свободные неизвестные равными нулю:

$$\begin{aligned} x_{r+1} = x_{r+2} = \dots = x_n &= 0. \\ x_1 = b'_1, \quad x_2 = b'_2, \dots, x_r &= b'_r. \end{aligned}$$

Полученное таким образом допустимое решение

$$(b'_1, b'_2, \dots, b'_r, 0, \dots, 0)$$

отвечает базису $x_1, x_2, \dots, x_r$, т.е. является базисным решением. Допустим для определенности, что мы ищем минимум f . Теперь нужно от данного базиса перейти к другому с таким расчетом, чтобы значение линейной функции f при этом уменьшилось. Проследим идею симплекс-метода на примере.

Пример 1. Дана система ограничений

$$\begin{cases} x_1 - 3x_2 + 5x_3 - x_4 = 2 \\ x_1 + x_2 + x_3 + x_4 = 4 \end{cases}$$

Требуется минимизировать линейную функцию $f = x_2 - x_3$. В качестве свободных переменных выберем x_2 и x_3 . Тогда данная система ограничений преобразуется к виду

$$\begin{cases} x_1 = 3 + x_2 - 3x_3, \\ x_4 = 1 - 2x_2 + 2x_3. \end{cases}$$

Таким образом, базисное решение (3, 0, 0, 1). Так как линейная функция уже записана в свободных неизвестных, то ее значение для данного базисного решения $f = 0$. Для уменьшения этого значения можно уменьшить x_2 или увеличить x_3 . Но x_2 в данном базисе равно нулю и потому его уменьшать нельзя. Попробуем увеличить x_3 . Первое из уравнений имеет ограничение $x_3 = 1$ (из условия $x_1 \geq 0$), второе - не дает ограничений. Далее, берем $x_3 = 1$, x_2 не меняем и получаем новое допустимое решение (0, 0, 1, 3), для которого $f = -1$ - уменьшилось. Найдем базис, которому соответствует это решение (он состоит, очевидно, из переменных x_3, x_4). От предыдущей системы ограничений переходим к новой:

$$\begin{cases} x_3 = 1 - \frac{1}{3}x_1 + \frac{1}{3}x_2, \\ x_4 = 3 - \frac{2}{3}x_1 + \frac{4}{3}x_2, \end{cases}$$

а форма в новых свободных переменных имеет вид

$$f = -1 + \frac{1}{3}x_1 + \frac{2}{3}x_2.$$

Теперь попробуем повторить предыдущую процедуру. Для уменьшения f надо уменьшить либо x_1 , либо x_2 , но это невозможно, так как в этом базисе $x_1 = 0, x_2 = 0$.

Таким образом, данное базисное решение является оптимальным, и $\min f = -1$ при $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 3$.

Приведем алгоритм симплекс-метода в общем виде. Обычно все вычисления по симплекс-методу сводят в стандартные таблицы.

Запишем систему ограничений в виде

$$\begin{cases} x_1 + a_{1,r+1}x_{r+1} + \dots + a_{1j}x_j + \dots + a_{1n}x_n = b_1, \\ \dots \\ x_i + a_{i,r+1}x_{r+1} + \dots + a_{ij}x_j + \dots + a_{in}x_n = b_i, \\ \dots \\ x_r + a_{r,r+1}x_{r+1} + \dots + a_{rj}x_j + \dots + a_{rn}x_n = b_r, \end{cases} \quad (7.90)$$

а функцию f

$$f + \gamma_{r+1}x_{r+1} + \dots + \gamma_jx_j + \dots + \gamma_nx_n = \gamma_0. \quad (7.91)$$

Тогда очередной шаг симплекс-процесса будет состоять в переходе от старого базиса к новому таким образом, чтобы значение линейной функции, по крайней мере, не увеличивалось.

Данные о коэффициентах уравнений и линейной функции занесем в табл. 7.12.

Таблица 7.12
Симплекс-таблица

Базис	Св. чл.	x_1	...	x_i	...	x_r	x_{r+1}	...	x_j	...	x_n
x_1	b_1	1	...	0	...	0	$a_{1,r+1}$	...	a_{1j}	...	a_{1n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
x_i	b_i	0	...	1	...	0	$a_{i,r+1}$	...	a_{ij}	...	a_{in}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
x_r	b_r	0	...	0	...	1	$a_{r,r+1}$	...	...	...	a_{rn}
f	γ_0	0	...	0	...	0	γ_{r+1}	...	γ_j	...	γ_n

Сформулируем алгоритм симплекс-метода применительно к данным, внесенным в табл. 7.12.

1. Выяснить, имеются ли в последней строке таблицы положительные числа (γ_0 не принимается во внимание). Если все числа отрицательны, то процесс закончен; базисное решение ($b_1, b_2, \dots, b_r, 0, \dots, 0$) является оптимальным; соответствующее значение целевой функции $f = \gamma_0$. Если в последней строке имеются положительные числа, перейти к п.2.

2. Просмотреть столбец, соответствующий положительному числу из последней строки, и выяснить, имеются ли в нем положительные числа. Если ни в одном из таких столбцов положительных чисел нет, то оптимального решения не существует. Если найден столбец, содержащий хотя бы один положительный элемент (если таких столбцов несколько, взять любой из них), отметить этот столбец и перейти к п. 3.

3. Разделить свободные члены на соответствующие положительные числа из выделенного столбца и выбрать наименьшее частное. Отметить строку таблицы, соответствующую наименьшему частному. Выделить разрешающий элемент, стоящий на пересечении отмеченных строки и столбца. Перейти к п. 4.

4. Разделить элементы выделенной строки исходной таблицы на разрешающий элемент (на месте разрешающего элемента появится единица). Полученная таким образом новая строка пишется на месте прежней в новой таблице. Перейти к п. 5.

5. Каждая следующая строка новой таблицы образуется сложением соответствующей строки исходной таблицы и строки, записанной в п. 4. которая предварительно умножается на такое число, чтобы в клетках выделенного столбца при сложении появились нули. На этом процесс заполнения новой таблицы заканчивается, и происходит переход к п. 1.

Таким образом, используя алгоритм симплекс-метода применительно к симплекс-таблице, мы можем найти оптимальное решение или показать, что его не существует. Результативность симплекс-метода гарантируется следующей теоремой (приведем ее без доказательства): *если существует оптимальное решение задачи линейного программирования, то существует и базисное оптимальное решение. Это решение может быть получено через конечное число шагов симплекс-методом, причем начинать можно с любого исходного базиса.*

Ранее мы предполагали, что если система ограничений задана в виде (7.85), то перед первым шагом она уже приведена к виду (7.86), где $b_i \geq 0$ ($i = 1, 2, \dots, r$). Последнее условие необходимо для использования симплекс-метода. Рассмотрим вопрос об отыскании начального базиса.

Один из методов его получения - метод симплексного преобразования.

Прежде всего проверяем, есть ли среди свободных членов отрицательные. Если свободные члены не являются числами неотрицательными, то добиться их неотрицательности можно несколькими способами:

1) умножить уравнения, содержащие отрицательные свободные члены, на -1;

2) найти среди уравнений, содержащих отрицательные свободные члены, уравнение с максимальным по абсолютной величине отрицательным свободным членом и затем сложить это уравнение со всеми остальными, содержащими отрицательные свободные члены, предварительно умножив его на -1.

Затем, используя действия, аналогичные указанным в пп. 3 - 5 алгоритма симплекс-метода, совершаем преобразования исходной таблицы до тех пор, пока не получим неотрицательное базисное решение.

Пример 2. Найти исходное неотрицательное базисное решение системы ограничений

$$\begin{cases} -5,871x_2 + 6,543x_3 - 9,996x_4 + 7,618x_5 = 0,864, \\ -8,704x_1 + 7,453x_2 + 8,962x_3 + 0,763x_4 - 8,654x_5 = 0, \\ 8,654x_1 - 6,729x_2 + 8,453x_3 + 7,772x_4 + 5,432x_5 = 9,743. \end{cases}$$

Так как условие неотрицательности свободных членов соблюдается, приступим к преобразованиям исходной системы, записывая результаты в таблицу. Согласно алгоритму просматриваем первый столбец. В этом столбце имеется единственный положительный элемент a_{31} . Делим на 8,654 все коэффициенты и свободный член третьей строки, после чего умножаем каждый коэффициент на 8,704 и складываем с соответствующими коэффициентами второй строки. Первая строка преобразований не требует, так как коэффициент при неизвестном x_1 равен нулю. В результате получаем

0,00000	-5,87100	6,54300	-9,99600	7,61800	0,86400
0,00000	0,68512	17,46384	8,57990	-3,19062	9,79929
1,00000	-0,77756	0,97677	0,89808	0,62769	1,11584

Продолжая просматривать второй столбец и совершая аналогичные преобразования, имеем

0,00000	0,00000	156,19554	63,52761	-19,72328	84,83688
0,00000	1,00000	25,49013	12,52318	-4,65701	14,30299
1,00000	0,00000	20,79687	10,63560	-2,99341	12,24727

И, наконец, на третьем шаге находим исходный базис. Его образуют неизвестные x_1, x_2, x_3 . Неизвестные x_4, x_5 являются свободными:

0,00000	0,00000	1,00000	0,40672	-0,12627	0,54315
0,00000	1,00000	0,00000	2,15588	-1,43829	0,45815
1,00000	0,00000	0,00000	2,17713	-0,36733	0,95155

При решении задачи линейного программирования целесообразно использование компьютера. В этом случае можно составить программу, решающую задачу. Учитывая, что программирование довольно трудоемко, можно посоветовать воспользоваться для оформления результатов расчетов табличным процессором. Кроме того, если получившаяся модель задачи слишком громоздка, можно воспользоваться математическими пакетами, которые позволяют получить решение задачи линейного программирования. И, наконец, еще один возможный вариант применения компьютеров - комбинирование всех вышеуказанных способов.

Контрольные вопросы и задания

1. Приведите примеры задач, приводящих к общей постановке задачи линейного программирования.
2. Сформулируйте задачу линейного программирования.
3. Сколько решений может иметь задача линейного программирования?
4. По каким причинам может отсутствовать решение задачи линейного программирования?
5. Каким образом неравенства из системы ограничений можно заменить уравнениями? Как задачу отыскания максимума линейной формы свести к задаче отыскания минимума?
6. Необходимо ли учитывать при записи решения дополнительные неизвестные, вводимые при переходе от неравенств к уравнениям?
7. Как найти начальный базис?
8. Сформулируйте алгоритм симплекс-метода.
9. Сформулируйте теорему о конечности алгоритма симплекс-метода.
10. Найдите максимум функции $z = 4x_1 + 3x_2$ ($x_i \geq 0$) при условии

$$\begin{cases} x_1 - x_2 \geq -2, \\ 5x_1 + 3x_2 \leq 15, \\ x_2 \leq 2,5, \\ 2x_1 - x_2 \geq -2, \\ x_1 - 2x_2 \leq 2. \end{cases}$$

11. Для откорма крупного рогатого скота используется два вида кормов b_1 и b_2 , в которые входят питательные вещества a_1, a_2, a_3 и a_4 . Содержание количеств единиц питательных веществ в одном килограмме каждого корма, стоимость одного килограмма корма и норма содержания питательных веществ в дневном рационе животного представлены в таблице. Составьте рацион при условии минимальной стоимости.

Питательные вещества	Вид кормов		Норма содержания питательного вещества
	b_1	b_2	
a_1	3	4	24
a_2	1	2	18

a_3	4	0	20
a_4	0	1	6
Стоимость 1 кг корма, руб.	2	1	

12. Трикотажная фабрика использует для производства свитеров и кофточек чистую шерсть, силон и нитрон, запасы которых составляют, соответственно, 800, 400 и 300 кг. Количество пряжи (кг), необходимое для изготовления 10 изделий, а также прибыль, получаемая от их реализации, приведены в таблице. Составьте план производства изделий, обеспечивающий получение максимальной прибыли.

Вид сырья в пряже	Затраты пряжи на 10 шт., кг	
	Свитер	Кофточка
Шерсть	4	2
Силон	2	1
Нитрон	1	1
Прибыль, руб.	6	5

13. При подкормке посевов необходимо внести на 1 га почвы не менее 8 единиц химического вещества A , не менее 21 единиц химического вещества B и не менее 16 единиц химического вещества C . Фермер закупает комбинированные удобрения двух видов I и II. В таблице указано содержание количества единиц химического вещества в 1 кг каждого вида удобрений и цена 1 кг удобрений. Определите потребность фермера в удобрениях I и II вида на 1 га посевной площади при минимальных затратах на их приобретение.

Химические вещества	Содержание химических веществ в 1 кг удобрения	
	I	II
A	1	5
B	12	3
C	4	4
Цена 1 кг удобрения, руб	5	2

14. На звероферме выращивают черно-бурых лис и песцов. Для обеспечения нормальных условий их выращивания используют три вида кормов. Количество корма каждого вида, которое должны получать животные, приведено в таблице. В ней также указаны общее количество корма каждого вида, которое может быть использовано зверофермой, и прибыль от реализации одной шкурки лисицы и песца. Составьте план выращивания животных, обеспечивающий максимальную прибыль.

Вид корма	Количество единиц корма, которое ежедневно должны получать		Общее количество корма
	Лисица	Песец	
1	2	3	180
11	4	1	240
III	6	7	426

Прибыль от реализации одной шкурки, руб	160	120	

Дополнительная литература к главе 7

1. *Араманович И. Г., Левин В. И.* Уравнения математической физики. - М.: Наука, 1969.
2. *Акулич И. Л.* Математическое программирование в примерах и задачах. - М.: Высшая школа, 1993.
3. *Беллман Р.* Математические методы в медицине: Пер. с англ. - М.: Мир, 1987.
4. *Белошапка В К* Информационное моделирование в примерах и задачах. -Омск: Из-во ОГПИ, 1992.
5. *Бигон М., Харпер Дж., Таунсенд К.* Экология. Особи, популяции и сообщества: Пер. с англ. В двух книгах. Кн. 1. - М.: Мир, 1989.
6. *Воеводин В. В.* Численные методы алгебры. - М.: Наука, 1966.
7. *Гнеденко Б. В.* Курс теории вероятностей. - М.: Наука, 1965.
8. *Гнеденко Б. В., Коваленко И. Н.* Введение в теорию массового обслуживания. -М.: Наука, 1966.
9. *Горстко А. Б.* Познакомьтесь с математическим моделированием. - М.: Знание, 1991.
10. *Горстко А. Б., Угольницкий Г. А.* Введение в моделирование эколого-экономических систем. - Ростов: Из-во РГУ, 1990.
11. *Гулд Х., Тобочник Я.* Компьютерное моделирование в физике: Пер. с англ. Т. 1, 2.-М.: Мир, 1990.
12. *Демидович Б. П., Марон И. А.* Основы вычислительной математики. - М.: Наука, 1970.
13. *Заварыкин В.М., Житомирский В. Г., Лапчик М.П.* Численные методы. - М.: Просвещение, 1990.
14. *Зайденберг А. П., Павлович Е.С.* Законы распределения случайных величин. -Омск: Изд-во ОГПИ, 1971.
15. *Зуховицкий С.И., Авдеева Л. И.* Линейное и выпуклое программирование. - М.: Наука, 1967.
16. *Кондаков В.М.* Математическое программирование. Элементы линейной алгебры и линейного программирования - Пермь: Из-во ПГУ, 1992.
17. *Марчук Г.И.* Математическое моделирование в иммунологии. - М.: Наука, 1991.
18. Математическое моделирование: Пер. с англ. / Под ред. Дж. Эндрюса, Р. Мак-Лоуна. - М.: Мир, 1979.
19. *Матюшкин-Герке А.* Учебно-прикладные задачи в курсе информатики. Информатика и образование, № 3-4, 5-6, 1992.
20. *Мигулин В. В., Медведев В. И., Мустель Е.Р., Парыгин В.Н.* Основы теории колебаний. - М.: Наука, 1988.
21. *Риклефс Р.* Основы общей экологии: Пер. с англ. - М.: Мир, 1979.
22. *Саати Т.* Элементы теории массового обслуживания и ее приложения: Пер. с англ. - М.: Сов. радио, 1971.
23. *Сайдашев А. А., Хеннер Е.К.* Компьютер на уроке математики. - Пермь: Из-во ПГУ, 1991.
24. *Самарский А. А., Гулин А.В.* Численные методы. - М.: Наука, 1989.
25. *Стрелков С П.* Введение в теорию колебаний. - М.: Наука, 1964.
26. *Шеннон Р.* Имитационное моделирование систем - искусство и наука: Пер. с англ. - М.: Мир, 1978.
27. Электронные вычислительные машины. / Под ред. А.Я.Соловьева. В 8 книгах. Книга 8. Решение прикладных задач. - М.: Высшая школа, 1987.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ	1
ВВЕДЕНИЕ.....	2
ЧАСТЬ ПЕРВАЯ	5
ГЛАВА 1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ.....	5
ВВЕДЕНИЕ.....	5
§ 1. ИНФОРМАТИКА КАК НАУКА И КАК ВИД ПРАКТИЧЕСКОЙ ДЕЯТЕЛЬНОСТИ	5
1.1. ИСТОРИЯ РАЗВИТИЯ ИНФОРМАТИКИ	5
1.2. ИНФОРМАТИКА КАК ЕДИНСТВО НАУКИ И ТЕХНОЛОГИИ.....	7
1.3. СТРУКТУРА СОВРЕМЕННОЙ ИНФОРМАТИКИ.....	8
1.4. МЕСТО ИНФОРМАТИКИ В СИСТЕМЕ НАУК.....	9
1.5. СОЦИАЛЬНЫЕ АСПЕКТЫ ИНФОРМАТИКИ	10
1.6. ПРАВОВЫЕ АСПЕКТЫ ИНФОРМАТИКИ	12
1.7. ЭТИЧЕСКИЕ АСПЕКТЫ ИНФОРМАТИКИ	14
<i>Контрольные вопросы</i>	15
§ 2. ИНФОРМАЦИЯ, ЕЕ ВИДЫ И СВОЙСТВА.....	15
2.1. РАЗЛИЧНЫЕ УРОВНИ ПРЕДСТАВЛЕНИЙ ОБ ИНФОРМАЦИИ.....	15
2.2. НЕПРЕРЫВНАЯ И ДИСКРЕТНАЯ ИНФОРМАЦИЯ	16
2.3. ЕДИНИЦЫ КОЛИЧЕСТВА ИНФОРМАЦИИ: ВЕРОЯТНОСТНЫЙ И ОБЪЕМНЫЙ	
ПОДХОДЫ	18
2.4. ИНФОРМАЦИЯ: БОЛЕЕ ШИРОКИЙ ВЗГЛЯД.....	21
2.5. ИНФОРМАЦИЯ И ФИЗИЧЕСКИЙ МИР	22
§ 3. СИСТЕМЫ СЧИСЛЕНИЯ	23
3.1. ПОЗИЦИОННЫЕ СИСТЕМЫ СЧИСЛЕНИЯ	23
3.2. ДВОИЧНАЯ СИСТЕМА СЧИСЛЕНИЯ.....	25
3.3. ВОСЬМЕРИЧНАЯ И ШЕСТНАДЦАТИРИЧНАЯ СИСТЕМЫ СЧИСЛЕНИЯ.....	26
§ 4. КОДИРОВАНИЕ ИНФОРМАЦИИ.....	28
4.1. АБСТРАКТНЫЙ АЛФАВИТ.....	28
4.2. КОДИРОВАНИЕ И ДЕКОДИРОВАНИЕ.....	29
4.3. ПОНЯТИЕ О ТЕОРЕМАХ ШЕННОНА	30
4.4. МЕЖДУНАРОДНЫЕ СИСТЕМЫ БАЙТОВОГО КОДИРОВАНИЯ	32
§ 5. ЭЛЕМЕНТЫ ТЕОРИИ ГРАФОВ	34
5.1. ОСНОВНЫЕ ПОНЯТИЯ.....	34
5.2. ПРЕДСТАВЛЕНИЕ ГРАФОВ	35
§ 6. АЛГОРИТМ И ЕГО СВОЙСТВА.....	36
6.1. РАЗЛИЧНЫЕ ПОДХОДЫ К ПОНЯТИЮ «АЛГОРИТМ»	36
6.2. ПОНЯТИЕ ИСПОЛНИТЕЛЯ АЛГОРИТМА	36
6.3. ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ АЛГОРИТМОВ	37
6.4. СВОЙСТВА АЛГОРИТМОВ.....	39
6.5. ПОНЯТИЕ АЛГОРИТМИЧЕСКОГО ЯЗЫКА	40
<i>Контрольные вопросы</i>	42
§ 7. ФОРМАЛИЗАЦИЯ ПОНЯТИЯ «АЛГОРИТМ»	42
7.1. ПОСТАНОВКА ПРОБЛЕМЫ.....	42
7.2. МАШИНА ПОСТА	43
7.3. МАШИНА ТЬЮРИНГА.....	47
7.4. НОРМАЛЬНЫЕ АЛГОРИТМЫ МАРКОВА.....	50
7.5. РЕКУРСИВНЫЕ ФУНКЦИИ	53
<i>Контрольные вопросы и задания</i>	55
§ 8. ПРИНЦИПЫ РАЗРАБОТКИ АЛГОРИТМОВ И ПРОГРАММ ДЛЯ РЕШЕНИЯ	
ПРИКЛАДНЫХ ЗАДАЧ.....	55
8.1. ОПЕРАЦИОНАЛЬНЫЙ ПОДХОД.....	55
8.2. СТРУКТУРНЫЙ ПОДХОД	58
8.3. НОВЕЙШИЕ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММ ДЛЯ ЭВМ	63
<i>Контрольные вопросы и задания</i>	64
§ 9. СТРУКТУРЫ ДАННЫХ	64
9.1. ДАННЫЕ И ИХ ОБРАБОТКА	64
9.2. ПРОСТЫЕ (НЕСТРУКТУРИРОВАННЫЕ) ТИПЫ ДАННЫХ	65
9.3. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ.....	66

Контрольные вопросы и задания	73
§ 10. ПОНЯТИЕ ОБ ИНФОРМАЦИОННОМ МОДЕЛИРОВАНИИ	73
10.1. МОДЕЛИРОВАНИЕ КАК МЕТОД РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ	73
10.2. ОСНОВНЫЕ ПОНЯТИЯ ИНФОРМАЦИОННОГО МОДЕЛИРОВАНИЯ	74
10.3. СВЯЗИ МЕЖДУ ОБЪЕКТАМИ	76
Контрольные вопросы и задания	78
§ 11. НЕКОТОРЫЕ КИБЕРНЕТИЧЕСКИЕ АСПЕКТЫ ИНФОРМАТИКИ	78
11.1. ПРЕДМЕТ КИБЕРНЕТИКИ.....	79
11.2. УПРАВЛЯЕМЫЕ СИСТЕМЫ	80
11.3. ФУНКЦИИ ЧЕЛОВЕКА И МАШИНЫ В СИСТЕМАХ УПРАВЛЕНИЯ	81
Контрольные вопросы и задания	82
§ 12. ПОНЯТИЕ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА.....	82
12.1. НАПРАВЛЕНИЯ ИССЛЕДОВАНИЙ И РАЗРАБОТОК В ОБЛАСТИ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА	82
12.2. ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В СИСТЕМАХ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА	84
12.3. МОДЕЛИРОВАНИЕ РАССУЖДЕНИЙ	86
12.4. ИНТЕЛЛЕКТУАЛЬНЫЙ ИНТЕРФЕЙС ИНФОРМАЦИОННОЙ СИСТЕМЫ	90
12.5. СТРУКТУРА СОВРЕМЕННОЙ СИСТЕМЫ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ.....	91
Контрольные вопросы и задания	91
Дополнительная литература к главе 1	93
ГЛАВА 2 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ	95
ВВЕДЕНИЕ.....	95
§ 1. ОПЕРАЦИОННЫЕ СИСТЕМЫ	95
1.1. НАЗНАЧЕНИЕ И ОСНОВНЫЕ ФУНКЦИИ ОПЕРАЦИОННЫХ СИСТЕМ	95
1.2. ПОНЯТИЕ ФАЙЛОВОЙ СИСТЕМЫ	98
1.3. ОПЕРАЦИОННЫЕ СИСТЕМЫ ДЛЯ КОМПЬЮТЕРОВ ТИПА IBM PC	101
1.4. ОБОЛОЧКИ ОПЕРАЦИОННЫХ СИСТЕМ.....	107
Контрольные вопросы и задания	117
§ 2. ПОНЯТИЕ О СИСТЕМЕ ПРОГРАММИРОВАНИЯ.....	117
2.1. ОСНОВНЫЕ ФУНКЦИИ И КОМПОНЕНТЫ	117
2.2. ТРАНСЛЯЦИЯ ПРОГРАММ И СОПУТСТВУЮЩИЕ ПРОЦЕССЫ	120
Контрольные вопросы	122
§3. ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ОБЩЕГО НАЗНАЧЕНИЯ.....	122
3.1. КЛАССИФИКАЦИЯ.....	122
3.2. ИНСТРУМЕНТАЛЬНЫЕ ПРОГРАММНЫЕ СРЕДСТВА ОБЩЕГО НАЗНАЧЕНИЯ	123
3.3. ИНСТРУМЕНТАЛЬНЫЕ ПРОГРАММНЫЕ СРЕДСТВА СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ	124
3.4. ПРОГРАММНЫЕ СРЕДСТВА ПРОФЕССИОНАЛЬНОГО УРОВНЯ.....	125
3.5. ОРГАНИЗАЦИЯ «МЕНЮ» В ПРОГРАММНЫХ СИСТЕМАХ	125
Контрольные вопросы и задания	128
§ 4. СИСТЕМЫ ОБРАБОТКИ ТЕКСТОВ	128
4.1. ЭЛЕМЕНТЫ ИЗДАТЕЛЬСКОГО ДЕЛА	128
4.2. ТЕКСТОВЫЕ РЕДАКТОРЫ	129
4.3. ИЗДАТЕЛЬСКИЕ СИСТЕМЫ	131
§ 5. СИСТЕМЫ КОМПЬЮТЕРНОЙ ГРАФИКИ.....	138
5.1. ПРИНЦИПЫ ФОРМИРОВАНИЯ ИЗОБРАЖЕНИЙ НА ЭКРАНЕ	138
5.2. ИЗОБРАЗИТЕЛЬНАЯ ГРАФИКА.....	139
5.3. ГРАФИЧЕСКИЕ РЕДАКТОРЫ	144
5.4. ДЕЛОВАЯ ГРАФИКА	146
5.5. ИНЖЕНЕРНАЯ ГРАФИКА	147
5.6. НАУЧНАЯ ГРАФИКА	149
Контрольные вопросы и упражнения	150
§ 6. БАЗЫ ДАННЫХ И СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ.....	150
6.1. ПОНЯТИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ	150
6.2. ВИДЫ СТРУКТУР ДАННЫХ	151
6.3. ВИДЫ БАЗ ДАННЫХ	152
6.4. СОСТАВ И ФУНКЦИИ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ.....	154
6.5. ПРИМЕРЫ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ	157
Контрольные вопросы и задания	163
§ 7. ЭЛЕКТРОННЫЕ ТАБЛИЦЫ.....	163

7.1. НАЗНАЧЕНИЕ И ОСНОВНЫЕ ФУНКЦИИ ТАБЛИЧНЫХ ПРОЦЕССОРОВ	163
7.2. ЭЛЕКТРОННЫЕ ТАБЛИЦЫ SUPERCALC.....	165
7.3. ЭЛЕКТРОННЫЕ ТАБЛИЦЫ EXCEL	172
§8. ИНТЕГРИРОВАННЫЕ ПРОГРАММНЫЕ СРЕДСТВА.....	173
8.1. ПРИНЦИПЫ ПОСТРОЕНИЯ ИНТЕГРИРОВАННЫХ ПРОГРАММНЫХ СИСТЕМ	173
8.2. ИНТЕГРИРОВАННЫЙ ПАКЕТ MS-WORKS	174
§ 9. ЭКСПЕРТНЫЕ СИСТЕМЫ	178
Контрольные вопросы и задания.....	183
§ 10. ИНСТРУМЕНТАЛЬНЫЕ ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ МАТЕМАТИЧЕСКИХ ЗАДАЧ.....	183
10.1. НАЗНАЧЕНИЕ ПРОГРАММ.....	183
10.2. ПАКЕТ МАТНСКАД	183
10.3. СИСТЕМА АНАЛИТИЧЕСКИХ ПРЕОБРАЗОВАНИЙ REDUCE	187
§ 11. КОМПЬЮТЕРНОЕ ТЕСТИРОВАНИЕ	190
11.1. ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ КОМПЬЮТЕРНЫХ ТЕСТОВ ПРЕДМЕТНОЙ ОБЛАСТИ	190
11.2. ТИПЫ КОМПЬЮТЕРНЫХ ТЕСТОВ.....	192
11.3. ИНСТРУМЕНТАЛЬНЫЕ ТЕСТОВЫЕ ОБОЛОЧКИ.....	193
11.4. ПРИМЕР ТЕСТА ПО ШКОЛЬНОМУ КУРСУ ИНФОРМАТИКИ.....	194
§12. КОМПЬЮТЕРНЫЕ ВИРУСЫ.....	197
12.1. ЧТО ТАКОЕ КОМПЬЮТЕРНЫЙ ВИРУС.....	197
12.2. РАЗНОВИДНОСТИ КОМПЬЮТЕРНЫХ ВИРУСОВ.....	198
12.3. АНТИВИРУСНЫЕ СРЕДСТВА	199
Контрольные вопросы и задания.....	200
§ 13. КОМПЬЮТЕРНЫЕ ИГРЫ.....	200
13.1. ВИДЫ И НАЗНАЧЕНИЕ КОМПЬЮТЕРНЫХ ИГР	200
13.1. ОБЗОР КОМПЬЮТЕРНЫХ ИГР	202
Контрольные вопросы	203
ГЛАВА 3 ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ	207
ВВЕДЕНИЕ.....	207
§ 1. ИСТОРИЯ РАЗВИТИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ.....	207
§2. ЯЗЫКИ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ	211
2.1. ПОНЯТИЕ О ЯЗЫКАХ ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ	211
2.2. МЕТАЯЗЫКИ ОПИСАНИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ.....	212
2.3. ГРАММАТИКА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ	213
§3. ПАСКАЛЬ КАК ЯЗЫК СТРУКТУРНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ	216
3.1. ВВЕДЕНИЕ.....	216
Контрольные вопросы	220
3.2. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА.....	220
Контрольные вопросы	223
3.3. СТРУКТУРЫ ДАННЫХ	223
3.4. ПРОЦЕДУРЫ И ФУНКЦИИ	234
3.5. РАБОТА С ФАЙЛАМИ.....	238
3.6. ДИНАМИЧЕСКИЕ ИНФОРМАЦИОННЫЕ СТРУКТУРЫ.....	243
Контрольные вопросы	247
3.7. РАБОТА С ГРАФИКОЙ.....	247
Контрольные вопросы и задания.....	253
3.8. ТУРБО-ОБОЛОЧКИ. ВЕРСИИ ПАСКАЛЯ.....	253
Контрольные вопросы	256
3.9. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЮ ТУРБО-ПАСКАЛЯ	256
§4. МЕТОДЫ И ИСКУССТВО ПРОГРАММИРОВАНИЯ	260
4.1. ПРОЕКТИРОВАНИЕ ПРОГРАММ	260
Контрольные вопросы и задания.....	265
4.2. ОСНОВНЫЕ ПРИНЦИПЫ РАЗРАБОТКИ И АНАЛИЗА АЛГОРИТМОВ	265
Задания.....	270
4.3. МЕТОДЫ ПОСТРОЕНИЯ АЛГОРИТМОВ, ОРИЕНТИРОВАННЫЕ НА СТРУКТУРЫ ДАННЫХ	270
Контрольные задания.....	273
4.4. РЕКУРСИВНЫЕ АЛГОРИТМЫ	273
Контрольные задания.....	277

4.5. ВАЖНЕЙШИЕ НЕВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ (ПОИСК И СОРТИРОВКА)	277
Контрольные вопросы и задания	288
§ 5. БЕЙСИК КАК ЯЗЫК ОПЕРАЦИОНАЛЬНО-ПРОБЛЕМНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ	288
5.1. ВВЕДЕНИЕ В БЕЙСИК	289
Контрольные вопросы	290
5.2. БАЗОВЫЕ ОПЕРАТОРЫ	290
Контрольные вопросы и задания	292
5.3. МУЗЫКАЛЬНЫЕ ВОЗМОЖНОСТИ	292
Контрольные вопросы и задания	293
5.4. ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ	293
Контрольные вопросы и задания	297
5.5. ОБРАБОТКА СИМВОЛЬНОЙ ИНФОРМАЦИИ	297
Контрольные вопросы и задания	299
5.6. ПОДПРОГРАММЫ	299
Контрольные вопросы	300
5.7. РАБОТА С ФАЙЛАМИ	300
5.8. СРЕДСТВА И МЕТОДЫ ОРГАНИЗАЦИИ ДИАЛОГА	302
Контрольные задания	303
5.9. ВЕРСИИ БЕЙСИКА	303
5.10. БЕЙСИК И ПАСКАЛЬ	304
§ 6. ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ СИ	304
6.1. ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКА И ПРИМЕР ПРОГРАММЫ НА СИ	304
6.2. ЭЛЕМЕНТЫ СИ: АЛФАВИТ, ИДЕНТИФИКАТОРЫ, ЛИТЕРАЛЫ, СЛУЖЕБНЫЕ СЛОВА	307
6.3. ТИПЫ ДАННЫХ И ОПЕРАЦИИ В ЯЗЫКЕ СИ. ВЫРАЖЕНИЯ	310
6.4. ОПЕРАТОРЫ. УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ ЯЗЫКА	315
6.5. СТРУКТУРА ПРОГРАММЫ НА СИ. ПОНЯТИЕ О ФУНКЦИЯХ	324
6.6. КЛАССЫ ПАМЯТИ	329
6.7. ФУНКЦИИ ВВОД-ВЫВОДА	332
6.8. ДИРЕКТИВЫ ПРЕПРОЦЕССОРА	336
6.9. СИ И ПАСКАЛЬ	338
§ 7. ОСНОВЫ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ ПРОЛОГ	339
7.1. ОБЩИЕ СВЕДЕНИЯ	339
7.2. АЛГОРИТМ ВЫПОЛНЕНИЯ ПРОГРАММ НА ПРОЛОГЕ	343
7.3. РЕКУРСИЯ	346
7.4. ПРЕДИКАТ ОТСЕЧЕНИЯ И УПРАВЛЕНИЕ ЛОГИЧЕСКИМ ВЫВОДОМ В ПРОГРАММАХ	348
7.5. ОБРАБОТКА СПИСКОВ	349
7.6. РЕШЕНИЕ ЛОГИЧЕСКИХ ЗАДАЧ НА ПРОЛОГЕ	352
Контрольные вопросы и задания	354
§ 8. ВВЕДЕНИЕ В ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ЛИСП	355
8.1. НАЗНАЧЕНИЕ И ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКА	355
8.2. ОСНОВНЫЕ ЭЛЕМЕНТЫ ПРОГРАММЫ НА ЛИСПЕ. СПИСКИ	356
8.3. ФУНКЦИИ	357
8.4. ФОРМЫ. УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ В ЛИСП-ПРОГРАММЕ	359
8.5. РЕКУРСИЯ И ЦИКЛ В ПРОГРАММАХ НА ЛИСПЕ	361
8.6. ВВОД-ВЫВОД ДАННЫХ	362
8.7. ПРИМЕР ПРОГРАММИРОВАНИЯ НА ЛИСПЕ	363
8.8. СВОЙСТВА СИМВОЛОВ	364
Контрольные вопросы и задания	364
§ 9. ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ	364
9.1. ОСНОВНЫЕ ПОЛОЖЕНИЯ	364
9.2. ОСНОВЫ ОБЪЕКТНОГО ПРОГРАММИРОВАНИЯ В СИСТЕМЕ ТУРБО-ПАСКАЛЬ	365
9.3. ОБОЛОЧКА TURBO-VISION	369
9.4. СРЕДА ОБЪЕКТНОГО ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ DELPHI	371
9.8. СИСТЕМА ОБЪЕКТНОГО ПРОГРАММИРОВАНИЯ SMALLTALK	374
Контрольные вопросы и задания	383
Дополнительная литература к главе 3	385
ЧАСТЬ ВТОРАЯ	387

ГЛАВА 4	ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА.....	387
	ВВЕДЕНИЕ.....	387
§ 1.	ИСТОРИЯ РАЗВИТИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ	387
1.1.	НАЧАЛЬНЫЙ ЭТАП РАЗВИТИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ	387
1.2.	НАЧАЛО СОВРЕМЕННОЙ ИСТОРИИ ЭЛЕКТРОННОЙ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ	389
1.3.	ПОКОЛЕНИЯ ЭВМ	391
1.4.	ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ	393
1.5.	И НЕ ТОЛЬКО ПЕРСОНАЛЬНЫЕ КОМПЬЮТЕРЫ... ..	394
1.6.	ЧТО ВПЕРЕДИ?	396
	<i>Контрольные вопросы</i>	396
§2.	АРХИТЕКТУРА ЭВМ	397
2.1.	О ПОНЯТИИ «АРХИТЕКТУРА ЭВМ»	397
2.2.	КЛАССИЧЕСКАЯ АРХИТЕКТУРА ЭВМ И ПРИНЦИПЫ ФОН НЕЙМАНА	398
2.3.	СОВЕРШЕНСТВОВАНИЕ И РАЗВИТИЕ ВНУТРЕННЕЙ СТРУКТУРЫ ЭВМ	400
2.4.	ОСНОВНОЙ ЦИКЛ РАБОТЫ ЭВМ	402
2.5.	СИСТЕМА КОМАНД ЭВМ И СПОСОБЫ ОБРАЩЕНИЯ К ДАННЫМ	403
	<i>Контрольные вопросы</i>	405
§3.	АРХИТЕКТУРА МИКРОПРОЦЕССОРОВ.....	406
3.1.	ИСТОРИЯ РАЗВИТИЯ МИКРОПРОЦЕССОРОВ	406
3.2.	ВНУТРЕННЯЯ ОРГАНИЗАЦИЯ МИКРОПРОЦЕССОРА	408
3.3.	РАБОТА МИКРОПРОЦЕССОРА С ПАМЯТЬЮ. МЕТОДЫ АДРЕСАЦИИ	410
3.4.	ФОРМАТЫ ДАННЫХ	412
3.5.	ОБРАБОТКА ПРЕРЫВАНИЙ	413
3.6.	РАБОТА МИКРОПРОЦЕССОРА С ВНЕШНИМИ УСТРОЙСТВАМИ.....	414
3.7.	ПРИМЕР: СИСТЕМА КОМАНД ПРОЦЕССОРОВ СЕМЕЙСТВА RDP	416
	<i>Контрольные вопросы и задания</i>	421
§4.	УЧЕБНАЯ МОДЕЛЬ МИКРОКОМПЬЮТЕРА	421
4.1.	СТРУКТУРА УЧЕБНОГО МИКРОКОМПЬЮТЕРА	422
4.2.	СИСТЕМА КОМАНД.....	424
4.3.	АДРЕСАЦИЯ ДАННЫХ	427
4.4.	РАБОТА С ВНЕШНИМИ УСТРОЙСТВАМИ	429
4.5.	ПРИМЕРЫ ПРОГРАММ.....	430
	<i>Контрольные вопросы и задания</i>	434
§ 5.	ВНЕШНИЕ УСТРОЙСТВА ЭВМ: ФИЗИЧЕСКИЕ ПРИНЦИПЫ И ХАРАКТЕРИСТИКИ.....	436
5.1.	ВНЕШНИЕ ЗАПОМИНАЮЩИЕ УСТРОЙСТВА.....	436
5.2.	УСТРОЙСТВА ВВОДА ИНФОРМАЦИИ	438
5.3.	УСТРОЙСТВА ВЫВОДА ИНФОРМАЦИИ	439
	<i>Контрольные вопросы и задания</i>	442
§ 6.	ЛОГИЧЕСКИЕ ОСНОВЫ ФУНКЦИОНИРОВАНИЯ ЭВМ	442
6.1.	ЛОГИКА ВЫСКАЗЫВАНИЙ. ЭЛЕМЕНТАРНЫЕ ЛОГИЧЕСКИЕ ФУНКЦИИ	442
6.2.	СХЕМНАЯ РЕАЛИЗАЦИЯ ЭЛЕМЕНТАРНЫХ ЛОГИЧЕСКИХ ОПЕРАЦИЙ. ТИПОВЫЕ ЛОГИЧЕСКИЕ УЗЛЫ.....	444
6.3.	ПРИМЕР ЭЛЕКТРОННОЙ РЕАЛИЗАЦИИ ЛОГИЧЕСКОГО ЭЛЕМЕНТА	449
	<i>Контрольные вопросы и задания</i>	450
	Дополнительная литература к главе 4.....	450
ГЛАВА 5	КОМПЬЮТЕРНЫЕ СЕТИ И ТЕЛЕКОММУНИКАЦИИ	451
	ВВЕДЕНИЕ.....	451
§ 1.	ЛОКАЛЬНЫЕ СЕТИ	452
1.1.	АППАРАТНЫЕ СРЕДСТВА	452
1.2.	КОНФИГУРАЦИИ ЛОКАЛЬНЫХ СЕТЕЙ И ОРГАНИЗАЦИЯ ОБМЕНА ИНФОРМАЦИЕЙ.....	454
1.3.	ЛОКАЛЬНЫЕ СЕТИ УЧЕБНОГО НАЗНАЧЕНИЯ	458
	<i>Контрольные вопросы</i>	459
§2.	ОПЕРАЦИОННЫЕ СИСТЕМЫ ЛОКАЛЬНЫХ СЕТЕЙ.....	460
	<i>Контрольные вопросы и задания</i>	465
§3.	ГЛОБАЛЬНЫЕ СЕТИ.....	465
3.1.	ОБЩИЕ ПРИНЦИПЫ ОРГАНИЗАЦИИ.....	465
3.2.	АППАРАТНЫЕ СРЕДСТВА И ПРОТОКОЛЫ ОБМЕНА ИНФОРМАЦИЕЙ	467
3.3.	ЭЛЕКТРОННАЯ ПОЧТА.....	469

3.4. ON-LINE INTERNET	477
§ 4. ПРЕДСТАВЛЕНИЕ ОБ ОПЕРАЦИОННОЙ СИСТЕМЕ UNIX	488
§ 5. ИСПОЛЬЗОВАНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ В ОБРАЗОВАНИИ.....	492
5.1. ТЕЛЕКОММУНИКАЦИИ КАК СРЕДСТВО ОБРАЗОВАТЕЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ	492
5.2. ПЕРСОНАЛЬНЫЙ ОБМЕН СООБЩЕНИЯМИ	494
5.3. ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ	495
5.4. СОВМЕСТНОЕ РЕШЕНИЕ ЗАДАЧ	496
ГЛАВА 6 ИНФОРМАЦИОННЫЕ СИСТЕМЫ	499
ВВЕДЕНИЕ	499
§ 1. БАНКИ ИНФОРМАЦИИ	500
1.1. БАНКИ ДАННЫХ.....	500
1.2. БАНКИ ДОКУМЕНТОВ.....	501
1.3. БАНК ПЕДАГОГИЧЕСКОЙ ИНФОРМАЦИИ	503
§ 2. БАЗЫ ДАННЫХ В СТРУКТУРЕ ИНФОРМАЦИОННЫХ СИСТЕМ.....	509
2.1. ОСНОВНЫЕ ПОНЯТИЯ	509
2.2. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ	511
2.3. ПРЕДСТАВЛЕНИЕ ОБ ЯЗЫКАХ УПРАВЛЕНИЯ РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ ТИПА dBASE	514
§ 3. АВТОМАТИЗИРОВАННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ	537
3.1. АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ	537
3.2. ИНФОРМАЦИОННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ	541
3.3. АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ НАУЧНЫХ ИССЛЕДОВАНИЙ.....	547
3.4. СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ.....	548
3.5. ГЕОИНФОРМАЦИОННЫЕ СИСТЕМЫ.....	548
Контрольные вопросы	550
§4. ЭКСПЕРТНЫЕ СИСТЕМЫ	550
Контрольные вопросы и задания	555
§ 5. КОМПЬЮТЕРНЫЕ ОБУЧАЮЩИЕ СИСТЕМЫ	555
5.1. ОСНОВНЫЕ ПРИНЦИПЫ НОВЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ОБУЧЕНИЯ.....	555
5.2. ТИПЫ ОБУЧАЮЩИХ ПРОГРАММ.....	558
5.3. КОМПЬЮТЕРНОЕ ТЕСТИРОВАНИЕ.....	564
5.4. ПЕРСПЕКТИВНЫЕ ИССЛЕДОВАНИЯ В ОБЛАСТИ КОМПЬЮТЕРНОГО ОБУЧЕНИЯ	571
ГЛАВА 7 КОМПЬЮТЕРНОЕ МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ	574
ВВЕДЕНИЕ	574
§ 1. О РАЗНОВИДНОСТЯХ МОДЕЛИРОВАНИЯ	575
§2. ПОНЯТИЕ О КОМПЬЮТЕРНОМ МАТЕМАТИЧЕСКОМ МОДЕЛИРОВАНИИ	576
2.1. МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ И КОМПЬЮТЕРЫ.....	576
2.2. ЭТАПЫ И ЦЕЛИ КОМПЬЮТЕРНОГО МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ	577
2.3. КЛАССИФИКАЦИЯ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ.....	580
2.4. НЕКОТОРЫЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ	581
§3. МОДЕЛИРОВАНИЕ ФИЗИЧЕСКИХ ПРОЦЕССОВ.....	590
3.1. ФИЗИКА И МОДЕЛИРОВАНИЕ.....	590
3.2. СВОБОДНОЕ ПАДЕНИЕ ТЕЛА С УЧЕТОМ СОПРОТИВЛЕНИЯ СРЕДЫ	591
3.3. ДВИЖЕНИЕ ТЕЛА, БРОШЕННОГО ПОД УГЛОМ К ГОРИЗОНТУ	598
ЗАКОНЫ ПОДОБИЯ	598
3.4. ДВИЖЕНИЕ ТЕЛА С ПЕРЕМЕННОЙ МАССОЙ: ВЗЛЕТ РАКЕТЫ	605
3.5. ДВИЖЕНИЕ НЕБЕСНЫХ ТЕЛ	607
3.6. ДВИЖЕНИЕ ЗАРЯЖЕННЫХ ЧАСТИЦ	610
3.7. КОЛЕБАНИЯ МАТЕМАТИЧЕСКОГО МАЯТНИКА.....	612
3.8. МОДЕЛИРОВАНИЕ ЯВЛЕНИЙ.....	620
И ПРОЦЕССОВ В ПРИБЛИЖЕНИИ СПЛОШНОЙ СРЕДЫ.....	620
3.9. МОДЕЛИРОВАНИЕ ПРОЦЕССА ТЕПЛОПРОВОДНОСТИ	627
Контрольные вопросы и задания	635
§ 4. КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В ЭКОЛОГИИ	637
4.1. ЭКОЛОГИЯ И МОДЕЛИРОВАНИЕ.....	638
4.2. МОДЕЛИ ВНУТРИВИДОВОЙ КОНКУРЕНЦИИ	639
4.3. ЛОГИСТИЧЕСКАЯ МОДЕЛЬ МЕЖВИДОВОЙ КОНКУРЕНЦИИ.....	644
4.4. ДИНАМИКА ЧИСЛЕННОСТИ ПОПУЛЯЦИЙ ХИЩНИКА И ЖЕРТВЫ.....	646

4.5. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ ДИНАМИКИ ПОПУЛЯЦИЙ.....	648
<i>Контрольные вопросы и задания</i>	652
§5. ГЛОБАЛЬНЫЕ МОДЕЛИ РАЗВИТИЯ ЧЕЛОВЕЧЕСТВА.....	653
§ 6. МОДЕЛИРОВАНИЕ СЛУЧАЙНЫХ ПРОЦЕССОВ.....	657
6.1. ТЕХНИКА СТОХАСТИЧЕСКОГО МОДЕЛИРОВАНИЯ.....	657
6.2. МОДЕЛИРОВАНИЕ СЛУЧАЙНЫХ ПРОЦЕССОВ В СИСТЕМАХ МАССОВОГО ОБСЛУЖИВАНИЯ.....	660
6.3. РАЗЛИЧНЫЕ ПРИМЕРЫ МОДЕЛИРОВАНИЯ СЛУЧАЙНЫХ ПРОЦЕССОВ.....	666
<i>Контрольные вопросы и задания</i>	670
§7. КОМПЬЮТЕРНОЕ МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ В ЭКОНОМИКЕ.....	672
7.1. ПОСТАНОВКА ЗАДАЧИ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ.....	672
7.2. СИМПЛЕКС-МЕТОД.....	677
<i>Контрольные вопросы и задания</i>	680
Дополнительная литература к главе 7.....	682
Содержание.....	683