Л. Пул

РАБОТА НА ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ

Работа на персональном компьютере

Using Your IBM Personal Computer

by Lon Poole

Howard W. Sams & Co., Inc.
1983

Л. Пул

РАБОТА НА ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ

Перевод с английского
канд. техн. наук **О. Ю. Горчинской**
под редакцией
канд. техн. наук **Е. К. Масловского**

Москва «Мир» 1986

ББК 32.97
П88
УДК 681.3

Пул Л.

П88 Работа на персональном компьютере: Пер. с англ.— М.: Мир, 1986.— 383 с., ил.

В книге американского специалиста рассмотрены методы работы на персональных компьютерах, снабженных необходимым числом периферийных устройств. Приведены конкретные примеры программирования на языке Бэйсик. Описаны способы ввода цифровой и графической информации с помощью дисплея, а также вывода информации на печатающие и громкоговорящие устройства. Много внимания уделено запуску, редактированию и завершению программ.

Для научных работников, инженеров, техников и студентов различных специальностей.

$\text{П}\frac{2405000000\text{-}423}{041\,(01)\text{-}86}$ 176-86, ч. 1 ББК 32.97

Редакция литературы по информатике и робототехнике

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

Персональные электронные вычислительные машины (ПВМ) призваны сыграть в жизни общества примерно ту же революционную роль, которую выполнил в свое время телефон, с той лишь разницей, что в данном случае речь идет не о всеобщей телефонизации, а о всеобщей компьютеризации. Персонализация вычислений с последующим объединением огромного количества персональных ЭВМ в единую вычислительную сеть — вот тот прогрессивный путь развития информационной технологии, который в шутку можно было бы обозначить девизом «Каждому — персональную машину!» Именно так можно будет справиться с пресловутой проблемой «информационного взрыва».

Считается, что ЭВМ может быть отнесена к разряду персональных, если она продается в сети розничной торговли (разумеется, как и пакеты программ), доступна по стоимости для семейного бюджета, оснащена хотя бы одним транслятором с широко используемого языка высокого уровня, содержит средства текстового и графического вывода информации, проста в обращении и рассчитана на массового неподготовленного пользователя. Иными словами, ПВМ во многом должна напоминать предмет домашнего обихода, подобный обычному холодильнику или телевизору. И в этом нет ничего невозможного: например, в США в 1985 году было продано около четырёх миллионов ПВМ, а годовой прирост объема сбыта в настоящее время составляет 50% и, видимо, будет расти.

Хотя в нашей стране число владельцев персональных ЭВМ пока невелико, подготовка к этапу, когда их будут миллионы, должна идти уже сейчас: в школах, высших и средних учебных заведениях, на производстве, на курсах повышения квалификации. И в этом плане предлагаемая в русском переводе книга американского автора Л. Пула «Работа на персональном компьютере», безусловно, сыграет свою положительную роль, так как в океане зарубежной литературы по персональным ЭВМ она привлекает доходчивостью изложения материала, конкретностью целей, практической направленностью и детальным рассмотрением наиболее важных вопросов использования ПВМ.

В оригинале название книги звучит буквально так: *Использование вашего персонального компьютера фирмы «Ай-Би-Эм»* (Using Your IBM Personal Computer), и придирчивый читатель сразу заметит неточность его перевода. Однако различия между ПВМ разных

фирм в аспекте их эксплуатации при наличии тенденции к «адаптации» становятся несущественными. Кроме того, книга, ориентированная на широкий круг неподготовленных читателей, должна быть понятна именно таким читателям уже с обложки.

Работая над переводом, мы старались сохранить присущую автору простоту изложения материала. Все, что требуется для чтения книги,— это понимание необходимости всеобщей «компьютерной грамотности», желание освоить «чудо-технику» и наличие возможности посидеть за дисплеем ЭВМ. Из области специальной терминологии достаточно лишь знать, что такое «дисплей», «дисковый накопитель», «пользователь» и «язык программирования».

В одной из глав автор предупреждает читателей о том, что при использовании помещенных в книге иллюстративных программ получаемые на экране «картинки» могут не совпадать с теми, что приведены в книге, поскольку в программах встречаются строки с бóльшим числом символов, чем это допускает ширина экрана используемого дисплея или домашнего телевизора в этом качестве. В русском переводе английские тексты, как известно, становятся длиннее, и поэтому авторское предупреждение остается в силе. Конечно, там, где это было возможно, мы старались «сжимать» перевод путем использования сокращений, но не в ущерб понятности комментариев, содержащихся в программах, и наглядности рисунков, играющих в книге очень важную роль. Впрочем, в ней вообще нет ничего лишнего, а многочисленные приложения могут рассматриваться как своего рода пособие по языку Бэйсик.

Было бы неправильно хоть как-то ограничивать круг возможных читателей этой книги — она для всех: у кого есть ПВМ, тот, несомненно, выигрывает в первую очередь, а «безмашинный» читатель после прочтения книги наверняка захочет приобрести ПВМ, как только это «разрешат» ему личный бюджет и предприятия-изготовители. В заключение позволим себе обратиться с призывом: «Папы и мамы! Увлекаясь электронными играми с вашей ПВМ, не гоните от машины детей! Они должны стать вашими учениками и превзойти своих учителей!»

Е. Масловский

ПРЕДИСЛОВИЕ

Эта книга предназначена для желающих пользоваться персональными электронными вычислительными машинами (ПВМ) фирмы IBM и программами, которые можно купить в готовом к применению виде или найти в журналах и книгах. Она несомненно представляет интерес и для тех, кто хочет научиться писать программы для ПВМ на языке Бэйсик независимо от того, вызвано ли такое желание простым любопытством или суровой необходимостью.

Книга охватывает широкий круг вопросов — от основ программирования до организации хранения данных в дисковой памяти и работы с графическими дисплеями. Она также дает представление о том, как пользоваться основными компонентами вычислительной системы: системными блоками, клавиатурой, экраном дисплея, печатающим устройством и накопителями на магнитных дисках; как обращаться с пакетом программ, купленным в записи на дискете или переписанным из книги либо журнала; как писать программы на языке Бэйсик.

Лучший способ научиться всему этому — самостоятельно выполнить рассматриваемые операции. Книга окажется еще более полезной, если вводить разбираемые примеры в собственную ПВМ. Поскольку события, происходящие во время выполнения тех или иных операций, описываются достаточно подробно, можно легко проверить получаемые результаты.

Книга состоит из двух частей, нескольких приложений и предметного указателя. Тем, кто хочет ограничиться приобретением навыков пользоваться коммерческими пакетами программ, достаточно ознакомиться с материалом ч. I. Кстати, это поможет успешно освоить материал ч. II, в которой основы программирования на языке Бэйсик изложены в расчете на лиц, никогда прежде не писавших программ для ЭВМ. С этой целью ч. II снабжена множеством таблиц, иллюстраций и примерами программ. Совместно с приложениями и указателем этот материал обеспечивает опытному программисту повседневную помощь в работе с языком Бэйсик ПВМ.

В главе 1 кратко перечисляются функции всех наиболее характерных компонентов ПВМ: клавиатуры, дисплея, системного блока, дисководов, печатающих устройств и программных средств. Подобно первой главе кулинарной книги, здесь приводятся сведения, предназначенные для тех, кто совершенно незнаком ни с компонентами ПВМ, ни с общей терминологией.

В главе 2 описываются процедуры включения вычислительной системы и работы с клавиатурой, сравниваются и противопоставляются различные типы дисплеев, даются общие рекомендации по использованию печатающего устройства.

Глава 3 посвящена накопителям на магнитных дисках (НМД): вопросам выбора и применения дискетов, работе дисковой операционной системы, а также организации дисковых файлов и использованию наиболее важных команд, обеспечивающих работу с дисками.

В главе 4 даются описания процедур использования дисковых пакетов программ, которые могут отличаться форматами команд, а также подробные указания по перезаписи программ с листингов в память ПВМ.

Глава 5 по существу является продолжением гл. 3: в ней содержатся некоторые дополнительные команды для работы с дисками и описывается ряд функциональных возможностей, которые могут оказаться полезными для опытного пользователя или программиста, работающего с языком Бэйсик. Рассматриваются эффективные способы редактирования команд, организации и автоматического выполнения потоков команд.

Глава 6 открывает вторую часть книги и знакомит читателя с основами языка Бэйсик. В ней показывается, как должны начинаться и заканчиваться Бэйсик-программы, рассматриваются способы написания простых программ для режима непосредственного взаимодействия и программируемого режима; описываются наиболее эффективные методы ввода и редактирования текстов программ, хранения и поиска Бэйсик-программ, размещенных на магнитных дисках.

В главе 7 описываются различные типы данных, с которыми может работать программа, написанная на языке Бэйсик, включая цепочки символов и три вида числовых значений. Объясняются аналогия и различие констант, переменных и массивов и представляется ряд команд, обеспечивающих манипулирование этими объектами и их совместное использование.

Глава 8 является продолжением гл. 7 и имеет дело с проблемами манипулирования данными. В ней представлены четыре разновидности выражений и способы их вычисления, объясняется, что такое функции и как они могут разрабатываться самим пользователем.

В главе 9 содержатся команды на языке Бэйсик, влияющие на порядок выполнения других команд программы: это команды ветвления, организации циклов, обращения к подпрограммам и использования перекрывающихся сегментов (оверлеев).

Глава 10 дает представление о том, как выводить строки символов и числовые значения на экран дисплея или на печатающее устройство, контролируя область их отображения и форму представления. Здесь же даются рекомендации по использованию конкретных функ-

циональных свойств и некоторых опций, присущих большинству печатающих устройств.

В главе 11 анализируются проблемы минимизации числа ошибок при вводе информации с клавиатуры за счет введения процедур общего контроля клавишного ввода и тщательной разработки операций ввода программ. Эти операции последовательно разворачиваются в стандартную процедуру, которой читатель может широко пользоваться в своей повседневной работе, связанной с программированием.

Глава 12 посвящена вопросам хранения и поиска данных на диске. Описывается общая структура дисковых файлов и два предусмотренных в языке Бэйсик метода доступа к данным — последовательный и произвольный. Изложение ведется на примере разработки весьма простой, но полезной прикладной программы учета личного имущества.

В главе 13 рассматриваются все специальные команды языка Бэйсик, предназначенные для формирования графических изображений. Из этой главы читатель узнает, как вычерчивать на экране дисплея линии, прямоугольники, круги, дуги, эллипсы и как их можно использовать для построения линейных графиков, гистограмм, круговых диаграмм и т. п. Проводится подробный разбор развитой системы команд машинной графики, которые значительно упрощают вычерчивание кривых сложной формы. Глава завершается разделом, посвященным машинной мультипликации, т. е. синтезу подвижных изображений; для большей наглядности изложения приводятся два практических примера.

Глава 14 касается вопросов синтеза звуковых образов; рассматриваются вопросы воспроизведения музыкальных сочинений при помощи динамика, имеющегося в комплекте ПВМ.

В главе 15 объясняется, как пользоваться предусмотренными в языке Бэйсик командами прямого управления наименее известными функциональными возможностями, такими, как установка цвета символов на экране графического дисплея, выбор одного из двух имеющихся в системе дисплеев и определение назначения функциональных клавиш.

В приложениях содержится полный набор команд языка Бэйсик, совокупность команд для работы с дисками и сообщения об ошибках; приводится перечень стандартных символов ПВМ в сопоставлении с символами клавиатуры большинства печатающих устройств.

В рамках каждой главы материал расположен по принципу возрастания сложности. Однако это отнюдь не означает, что материал, помещенный в конце гл. 4, проще, чем материал, содержащийся в начале гл. 6, поскольку главы организованы прежде всего по темам. Благодаря такой структуре книги можно свободно перейти к чтению следующей главы, если предшествующая ей глава оказалась слишком трудной для понимания. Но если подобное ощущение воз-

никнет при чтении последующих глав, то лучше вернуться к предыдущей и дополнительно поработать над изложенным в ней материалом.

Книга учит писать программы для ПВМ на языке Бэйсик, позволяющие наилучшим образом использовать наиболее известные возможности машины, однако при этом не преследуется цель подробно разъяснять смысл абсолютно всех команд языка Бэйсик. Во второй части рассмотрено около 80% команд и опций, доступных для реализации на ПВМ. В число остальных 20% команд, которые подробно не анализируются, входят команды и опции, относящиеся к редко используемым возможностям ПВМ (например, хранение и поиск программ и данных на кассетных магнитных лентах) и к более сложным процедурам, которыми пользуется сравнительно небольшое число программистов, работающих с языком Бэйсик (например, программирование на машинном языке).

Эта книга не увидела бы свет, если бы не усилия моих многочисленных коллег и друзей. В этой связи я хотел бы выразить свою признательность Нэнси Фишер за помощь, оказанную мне при написании гл. 14, и Эрферт Нильсон за подготовку рукописи. Я также признателен Янису Паско, руководителю отдела сбыта и развития фирмы Howard W. Sams & Co., которому я обязан тем, что книга была опубликована, д-ру Озборну за то, что именно он сделал мне предложение написать эту книгу, а также за его неоценимые советы и неиссякаемый оптимизм. Особой благодарности заслуживает мужественная Карин, которая стойко выносила мои проявления мономании, терпеливо выслушивала все мои предложения и давала правильную оценку событиям благодаря присущему ей здравому смыслу.

Корпорация AMDEK в лице Дэна Раймса любезно предоставила в мое распоряжение монитор AMDEK Color II, без которого я не смог бы со всей полнотой ощутить возможности графических средств персональной ЭВМ.

Часть материала гл. 1 и 13 была первоначально опубликована в журнале PC («The Independent Guide to IBM Personal Computers»). Идеи оживления изображений, изложенные в гл. 13, заимствованы из книги «Film Animation as a Hobby». (Andrew and Mark Hobson, Sterling Publishing Co., New York, 1975).

Часть I

РАБОТА НА ПЕРСОНАЛЬНОЙ ЭВМ

Глава 1

СТРУКТУРА ПЕРСОНАЛЬНОЙ ЭВМ

Персональная ЭВМ (ПВМ) — это сложная вычислительная система, каждая часть которой имеет свое функциональное назначение. Ядро ПВМ составляют клавиатура, системный блок и видеодисплей (рис. 1.1), обеспечивающие выполнение основных операций: ввод, обработку и выдачу информации. Для повышения эффективности этих операций используется широкий набор дополнительных средств, благодаря чему становится возможным хранение огромного количества информации, печатание различных отчетов, взаимодействие с другими ЭВМ и реализация множества других функций. В данной главе рассматриваются наиболее распространенные как обязательные, так и факультативные узлы ПВМ, выпускаемые фирмой IBM и другими изготовителями.

Клавиатура и видеодисплей

Клавиатура представляет собой стандартное устройство, предназначенное для ввода команд и данных. Используя 83 клавиши клавиатуры в различных сочетаниях, можно ввести любой из 256 символов, распознаваемых ПВМ.

Основным средством взаимодействия пользователя с любой ЭВМ является экран видеодисплея. На экране можно разместить до 25 строк текста, хотя использование нижней строки в определенном смысле ограничено: максимальная длина строки 40 или 80 символов. Для большинства экранов размеры знаков 40-символьной строки вдвое больше по сравнению с 80-символьной строкой. (Очевидно, что большие по размеру знаки воспринимаются легче.)

Видеодисплеи в ПВМ могут быть любого типа. В составе ПВМ фирмы IBM предусмотрен одноцветный (монохроматический) видеомонитор (рис. 1.1), воспроизводящий на экране текстовую информацию и ограниченный набор графических изображений в зеленом цвете на черном фоне. Аналогичные видеомониторы выпускают и другие фирмы, причем в одних изображение воспроизводится в белом цвете на черном фоне, а в других — зеленом на черном. Монохроматические мониторы, как правило, дают очень четкое изображение, поэтому их хорошо использовать для воспроизведения текстовой информации, состоящей из 40- или 80-символьных строк.

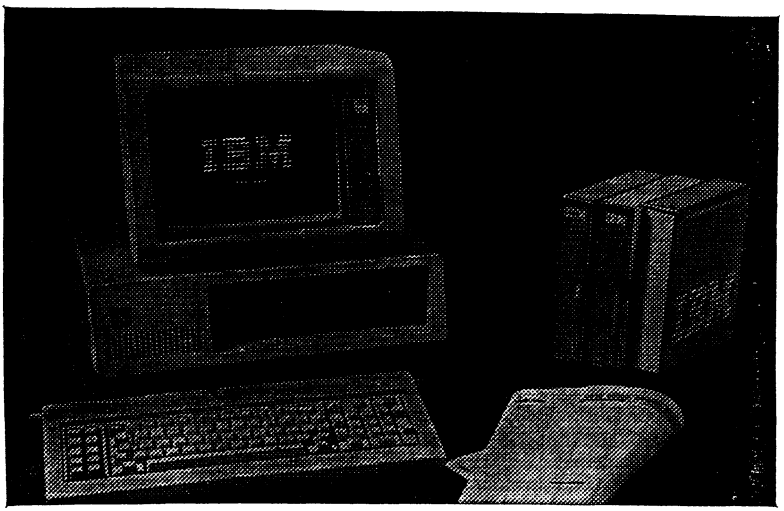


Рис. 1.1. Комплект персональной ЭВМ с монохроматическим видеомонитором.

Персональная ЭВМ способна воспроизводить текст и любые графические изображения в цвете, если в качестве видеодисплея используется, например, обычный (бытовой) телевизор, снабженный двумя дополнительными специальными устройствами (рис. 1.2), одно из которых представляет собой частотный модулятор, преобразующий видеосигнал ЭВМ в форму, воспринимаемую телевизором, а другое — обычный переключатель для настройки телевизора на прием сигнала либо от ЭВМ, либо с обычной антенны. Однако по своей разрешающей способности бытовые телевизоры не соответствуют возможностям ЭВМ, и даже в телевизорах самых лучших марок четкость изображения оказывается недостаточной для воспроизведения 80-символьных текстовых строк.

Несколько большей оптической разрешающей способностью по сравнению с лучшими бытовыми телевизорами обладают так называемые *комбинированные* мониторы. Более дорогие цветные мониторы (RGB) могут воспроизводить каждый бит файла оперативной информации, генерируемой ПВМ (рис. 1.3). Следует заметить, что ни в одном из цветных мониторов не применяются ни высокочастотный модулятор, ни антенный переключатель.

◆◆◆ Нельзя пользоваться одновременно монохроматическим монитором, например фирмы IBM, и каким-либо цветным монитором. Хотя вычислительная машина и может быть ориентирована на применение мониторов двух типов, назначение каждого из них должно быть определено заранее. Например, можно использовать монохро-

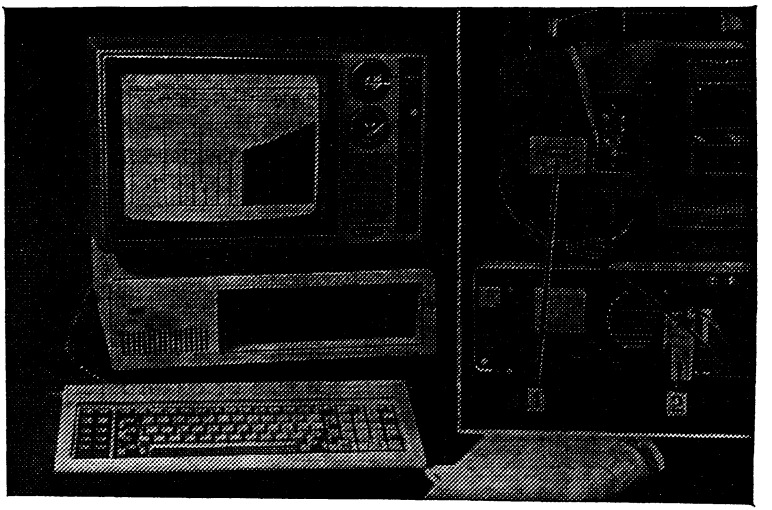


Рис. 1.2. Комплект ПВМ с бытовым телевизором в качестве видеомонитора.
1 — антенный переключатель; *2* — высокочастотный модулятор.

Рис. 1.3. Комплект ПВМ с трехцветным видеомонитором.

матический видеомонитор фирмы IBM для визуального отображения текста и бытовой цветной телевизор для отображения цветных графических данных, однако ПВМ не сможет обеспечить воспроизведение изображений на обоих дисплеях одновременно. Сложная процедура переключения ПВМ с монохроматического монитора фирмы IBM на цветной монитор и обратно описывается в гл. 15.

Системный блок

Основой всей конструкции персональной ЭВМ является системный блок, управляющий потоком информации между всеми имеющимися в нем платами других устройств. Этот блок позволяет не только направлять необработанные данные от одного устройства к другому, но и обрабатывать поступающую информацию: выбирать, комбинировать и исключать данные, выполнять расчеты для получения на этой основе совершенно иных видов информации.

Естественно предположить, что реализующий такие функции системный узел должен представлять собой весьма сложное устройство. Чтобы убедиться в том, что это действительно так, достаточно заглянуть внутрь блока (рис. 1.4). Однако, чтобы пользоваться ПВМ, необязательно разбираться во всех тонкостях ее внутренних схем, необходимо лишь знать возможности системного блока ПВМ. Пользование ПВМ подобно пользованию автомобилем. Владельцу машины известно, что под капотом автомобиля находятся двигатель, аккумулятор, вероятно, радиатор, возможно, кондиционер и еще многие другие устройства. Можно допустить, что он не знаком с принципом работы этих узлов (или это его просто не интересует), но о существовании таких узлов он, несомненно, знает и, следовательно, имеет представление о возможностях своей машины. Аналогичная ситуация наблюдается при работе на ПВМ, системный блок которой имеет источник питания, микропроцессор, оперативное запоминающее устройство, а может быть, и один или несколько дисководов. Некоторые внутренние устройства системного блока трудно обнаружить без его разборки. Так, например, системная плата обычно располагается в самом низу системного блока, и поэтому ее многочисленные интегральные схемы, в том числе микросхемы микропроцессора и запоминающего устройства, оказываются скрытыми от глаз.

Источник питания и микропроцессор

Управление потоком электроэнергии от источника питания системного блока, который в свою очередь подает напряжение соответствующего номинала на внутренние электронные схемы, приводы

дискет и клавиатуру, осуществляется с помощью основного выключателя системы. Для охлаждения электрических цепей системного блока на внутренней поверхности крышки источника питания установлен вентилятор.

Рядом с источником питания на системной плате расположен кристалл большой интегральной схемы (БИС), микропроцессор Intel 8088,— «мозг» всего системного блока, поскольку именно микропроцессор координирует выполняемые операции и производит арифметические вычисления, необходимые для работы системы в целом. Микропроцессор 8088 обладает довольно высоким быстродействием. При необходимости его скорость может быть увеличена путем добавления сопроцессора, плата которого вставляется в со-

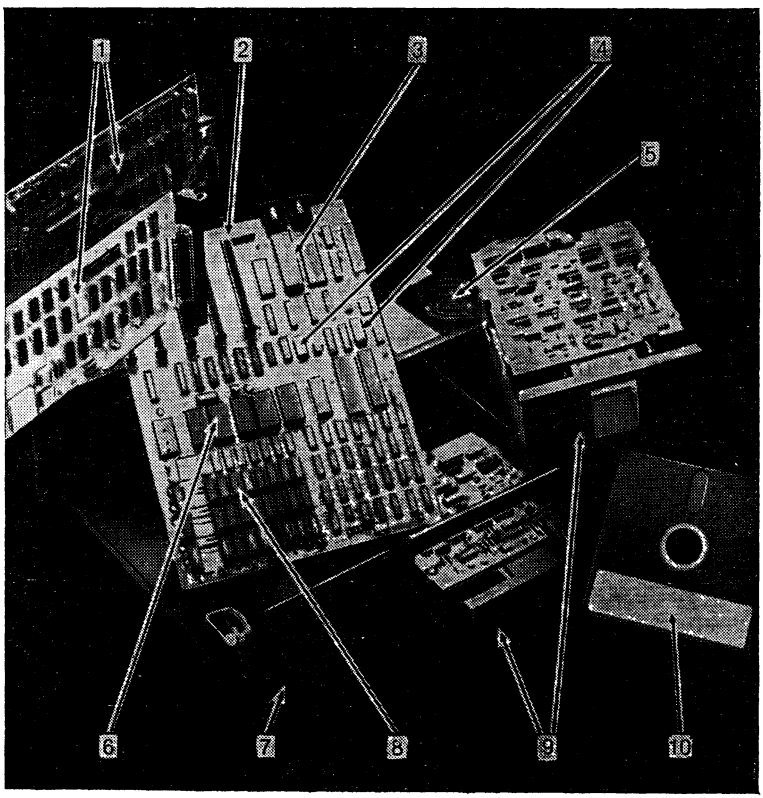


Рис. 1.4. Внутреннее устройство системного блока.

1 — адаптерные платы; *2* — гнезда для расширительных модулей; *3* — микропроцессор; *4* — переключатели выбора конфигурации; *5* — источники питания; *6* — постоянное запоминающее устройство; *7* — динамик; *8* — динамическая память; *9* — дисководы; *10* — дискет.

седнее пустое гнездо. В роли сопроцессоров могут выступать процессор числовых данных Intel 8087 и быстродействующий процессор ввода-вывода (ПВВ) Intel 8089.

Память

В комплекте каждой ПВМ имеется запоминающее устройство (ЗУ) определенной емкости, которая измеряется в единицах, называемых *байтами*. Один байт памяти ЭВМ — одна машинная ячейка — способен хранить только один символ, поэтому можно рассматривать байты как информационные символы, хотя в памяти ЭВМ помимо них хранится и другая информация, например числовые величины и программы. Благодаря определенным особенностям архитектуры ЭВМ емкость ЗУ обычно кратна 1024 байт. Эта единица емкости памяти называется *килобайтом* (1К). Персональная ЭВМ может иметь память емкостью от 60К (61 440 байт) до 1024К (1 048 576 байт). Емкость памяти иногда измеряется в мегабайтах: 1М байт=1024К байт.

Любая ПВМ оснащена постоянной (неизменяемой) памятью, называемой *постоянным запоминающим устройством* (ПЗУ); нужная информация записывается в ПЗУ емкостью 40К байт в нестираемой форме в процессе его изготовления и не изменяется даже при отключении источника питания. В ПЗУ хранится информация, важная для интерпретирования команд на языке Бэйсик и для выполнения других общих функций.

Имеется еще несколько типов ЗУ памяти ПВМ: динамическое ЗУ, память с оперативной записью и считыванием и ЗУ с произвольной выборкой (ЗУПВ). В ЗУПВ информация может храниться и извлекаться из него в любой момент работы ПВМ, но при отключении источника питания ее содержимое уничтожается. Главное достоинство динамической памяти — изменчивость, благодаря которой одно и то же запоминающее устройство можно многократно использовать для различных целей. На системной плате имеется по крайней мере 16К байт динамической памяти, которая может быть увеличена до 32, 48 или 64К байт путем установки дополнительных микросхем. Дальнейшее расширение динамической памяти может быть осуществлено за счет применения адаптерных плат.

Переключатели выбора конфигурации системы

Рядом с платой микропроцессора 8088 расположены две группы переключателей, установка которых в определенное положение задает нужную конфигурацию системы: требуемую емкость памяти, тип применяемого дисплея и т. п. Положение переключателей в зависимости от выбираемого варианта конфигурации вычислительной

системы определяется инструкциями по монтажу дополнительного оборудования, которые входят в комплект документации каждой ПВМ.

Встроенный динамик, расширительные гнезда и адаптерные платы

По диагонали от источника питания находится небольшой громкоговоритель для подачи звукового сигнала в самых различных ситуациях. Он может быть также использован в специальных программах для воспроизведения музыки и звуковых эффектов.

На тыльной стороне системной платы имеется пять гнезд, в которые вставляются адаптерные платы. При наличии более пяти адаптерных плат различного назначения количество предусмотренных гнезд оказывается недостаточным, и для увеличения их числа применяется специальный расширительный блок.

Адаптерные платы (рис. 1.5) делают ПВМ универсальной; некоторые из них обеспечивают работу внешнего оборудования (например, дисплеев, дисководов, печатающих устройств, электронных игр) и связь с другими ЭВМ. Часть адаптерных плат (платы дополнительной динамической памяти, электронных часов с батарейным питанием и тестеров экспериментальных схем) является автономной и представляет собой самостоятельные функциональные узлы. Существуют также отдельные адаптерные платы, объединяющие

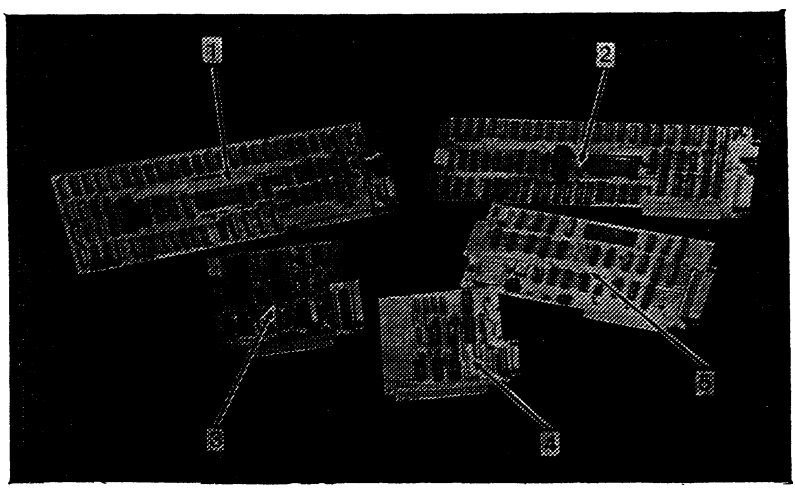


Рис. 1.5. Некоторые адаптерные платы.

1 — адаптер цветных и черно-белых графических устройств; *2* — адаптер монохроматического дисплея и параллельного печатающего устройства; *3* — адаптер асинхронной связи (стандарт RS232 для последовательной передачи); *4* — адаптер блока управления электронными играми; *5* — адаптер дискетных накопителей.

Таблица 1.1. **Дополнительная аппаратура и соответствующие адаптеры**

Возможные устройства и подключения	Требуемая адаптерная плата
Дополнительная память	Расширитель памяти
Непрерывный датчик времени/календарь	Часы с батарейным питанием
Цветной монитор	Адаптер цветных и черно-белых графических устройств
Дискеты	Адаптер дискетного накопителя
Экспериментальная схемная плата	Адаптер прототипа и расширителя
Бытовой телевизор	Адаптер цветных и черно-белых графических устройств
Рычаги управления	Адаптер блока управления электронными играми
Световое перо	Адаптер цветных и черно-белых графических устройств
Модем	Адаптер асинхронной связи
Монохроматический монитор	Адаптер монохроматического дисплея и параллельного печатающего устройства фирмы IBM
Сеть передачи данных	Адаптер асинхронной связи
Пульты электронных игр	Адаптер блока управления электронными играми
Печатающее устройство	Адаптер монохроматического дисплея и параллельного печатающего устройства фирмы IBM, адаптер параллельного печатающего устройства или адаптер асинхронной связи
Сеть связи	Адаптер асинхронной связи
Видеотекст	Адаптер асинхронной связи
Винчестерский диск	Адаптер винчестерских дисков

несколько функций, что обеспечивает экономию гнезд системного блока и создает возможность подключения большего количества дополнительных устройств без применения расширительного блока.

Тип адаптерных плат, необходимых для конкретной вычислительной системы, зависит от того, для каких целей применяется ПВМ и какие ее внешние компоненты используются. Некоторые возможные варианты перечислены в табл. 1.1.

Диски и дисковые накопители

Магнитный диск — дополнительное запоминающее устройство, своего рода реализация расширения памяти. На диске может постоянно храниться любое содержимое внутренней памяти ПВМ и многократно вызываться для повторного использования. Это означает, что ПВМ способна переключаться с одной задачи на другую со скоростью, определяемой временем обращения к программе на

магнитном диске, а длина и сложность программ уже не ограничиваются емкостью ЗУ, поскольку ненужную в данный момент для работы программы информацию можно записывать на диск и использовать освобождающуюся за счет этого динамическую память для оперативной информации, вызываемой с диска. В целом дисковый накопитель состоит из трех частей: адаптерной платы, дисковода и собственно диска (рис. 1.6). Адаптер координирует обмен информацией между микропроцессором и динамической памятью, с одной стороны, и НМД — с другой; он вставляется в одно из гнезд для расширительных модулей, находящихся внутри системного блока. Специальные устройства накопителя считывают и записывают информацию на диск, работая по принципу магнитной звуко- и видеозаписи. Собственно диск представляет собой жесткую пластинку, которая вращается внутри дисковода и имеет в качестве запоминающей среды слой магнитного покрытия.

Дискеты

В небольших вычислительных системах, подобных ПВМ, наиболее распространенным типом диска является *гибкий диск*, обычно называемый *дискетом*. Дискеты выпускаются двух размеров (по диаметру): $5^1/_4$ дюйма (133 мм) и 8 дюймов (203 мм) и представляют собой гибкий пластик в форме диска, покрытый магнитной пленкой и помещенный в твердый пластмассовый конверт для защиты от

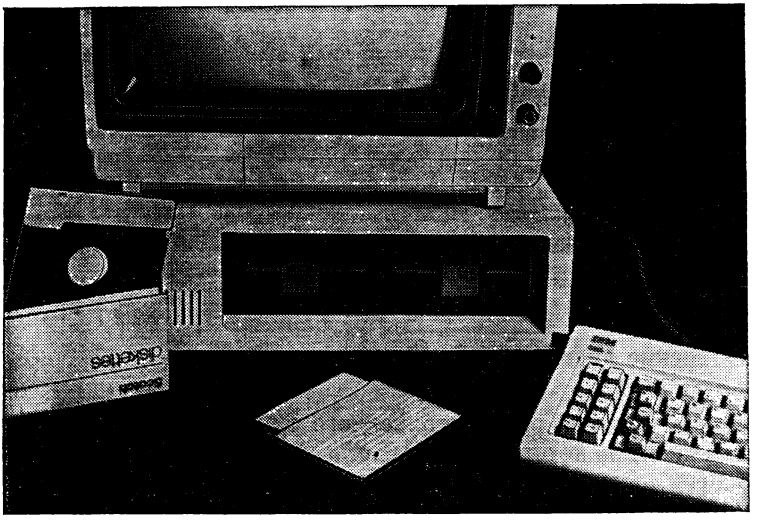


Рис. 1.6. Дискетные накопители и дискеты.

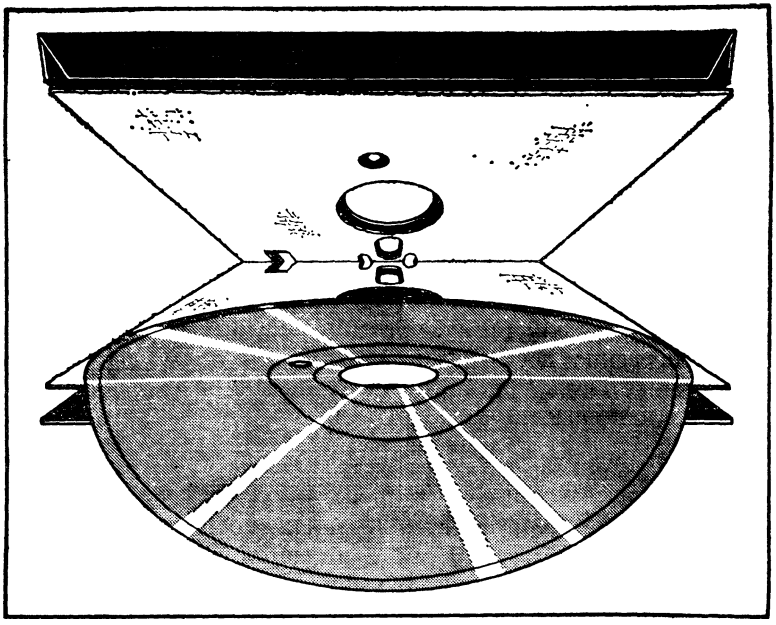


Рис. 1.7. Внутреннее устройство дискета.

физических повреждений (рис. 1.7). Емкость отдельного дискета колеблется от 90К байт до 1М байт и более в зависимости от применяемого дисковода.

Благодаря тому что дискет существует отдельно от дисковода, можно использовать несколько дискетов, каждый из которых содержит различную информацию и управляется одним и тем же приводом. Дискет вставляется в щелевое отверстие дисковода, механизм привода захватывает гибкий пластмассовый диск за центральное отверстие и вращает его внутри конверта. Доступ к информации осуществляется через прорези в защитном конверте.

Винчестерские диски

Основой другого распространенного типа дискового ЗУ служит жесткий диск, на который информация записывается по специальной технологии, позволяющей хранить на одной стороне диска в 100 раз больший объем данных, чем при использовании обычной технологии. Этот специальный способ записи называется *винчестерской технологией*, а ЗУ, в которых она используется, носят название *винчестер-*

ских дисков [1]. Диски такого типа чрезвычайно чувствительны даже к мельчайшим частичкам пыли или копоти, поэтому они составляют единое целое с приводом и не могут сниматься с дисковода. Сам диск заключен в герметизированный корпус и может заменяться только персоналом, обслуживающим ЭВМ. Емкость памяти ПВМ, в которых используются винчестерские диски, может превышать 5М байт.

Печатающие устройства

При наличии печатающего устройства ПВМ может печатать на бумаге отрывные документальные сообщения, сводки, отчеты и т. п. Существует целая гамма печатающих устройств для ПВМ (рис. 1.8),

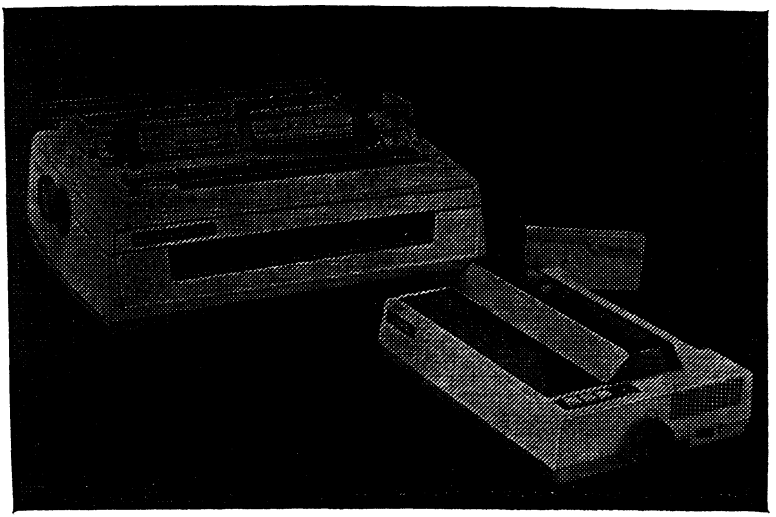


Рис. 1.8. Печатающие устройства и буфер печати.

которые отличаются друг от друга по скорости, качеству печати, набору воспроизводимых знаков, типу используемой бумаги и по ряду других характеристик.

[1] Изначально такие накопители рассчитывались на 2 диска по 30 Мбайт, составляющих единый блок; результирующая емкость получаемого НМД обозначалась цифрами 30/30, подобно калибру старинного охотничьего ружья «Винчестер». Отсюда и возникло название «винчестерский диск», или «винчестер».— *Прим. ред.*

Параллельный и последовательный режимы связи

Из всех известных режимов связи между ЭВМ и печатающими устройствами наиболее распространены два: параллельный и последовательный [1]. В ПВМ может реализовываться любой из двух указанных способов информационного обмена, но параллельный режим определен как стандартный.

В матричном печатающем устройстве IBM 80CPS реализован параллельный режим передачи данных; оно подключается либо к адаптерной плате монохроматического дисплея фирмы IBM и параллельного печатающего устройства, либо к адаптеру параллельного печатающего устройства фирмы IBM. Взамен последнего можно выбрать любой другой тип параллельной печати. Печатающие устройства последовательного действия соединяются с ПВМ через адаптер асинхронной связи фирмы IBM или подключаются к адаптерной плате какого-нибудь другого режима последовательной связи.

Буферы печатающих устройств

В процессе вывода информации на печать ПВМ большую часть времени простаивает, поскольку ее потенциальное быстродействие по выводу данных значительно превосходит скорость работы даже самого быстрого печатающего устройства. Для устранения этого несоответствия можно включить между ПВМ и печатающим устройством промежуточное аппаратное средство, которое будет играть роль своеобразного накопительного резервуара. Такое устройство (рис. 1.8), называемое *буфером печати* или *блоком подкачки печати*, оснащается (подобно системному блоку) динамической памятью, предназначенной для хранения информации, адресованной печатающему устройству. Буфер печати принимает информацию с высокой скоростью, запоминает ее и постепенно, по мере готовности печатающего устройства, выдает на печать. Такой буфер можно представить себе в виде ведра с отверстием в днище. Машина заполняет буфер информацией точно так же, как вода заполняет ведро, а из буфера информация «просачивается» на печатающее устройство, подобно воде, вытекающей из ведра через отверстие в дне.

Для того чтобы напечатать 16K знаков, печатающему устройству требуется от 2 до 5 мин, в то время как ПВМ способна заполнить буфер емкостью 16K менее чем за 1 мин. При заполнении буфера ПВМ должна снижать свое быстродействие до уровня скорости печати, чтобы добавить новую порцию информации по мере поступления ее из буфера на печатающее устройство. В конечном итоге ПВМ

[1] Строгие различия между ними носят скорее технический характер и совершенно несущественны в аспекте рассматриваемых в книге вопросов.

завершит заполнение буфера и перейдет к выполнению других операций, а печатание будет продолжаться автономно до тех пор, пока буфер полностью не очистится.

Система программного обеспечения

ПВМ без программы подобна оркестру без партитуры — она бездействует. Любой ЭВМ необходимы команды, которые вызвали бы ее к жизни, точно так же, как оркестру необходимо музыкальное произведение, записанное на бумаге в нотных знаках. Программа — это упорядоченная совокупность команд, сообщающих ЭВМ, как выполнять ту или иную операцию, и этот факт делает программы столь же важной частью ПВМ, как и любое физическое устройство. Набор программ, имеющихся в наличии у данной ЭВМ, называют *системой программного обеспечения*.

В памяти ПВМ одновременно сосуществуют несколько типов программ, которые служат единой цели: совместно управляют работой вычислительной системы. Один из типов программ определяет, для какого вида деятельности применяется ЭВМ: для обработки текстовой информации, выполнения бухгалтерских расчетов, финансового анализа, для работы с видеотексом, организации досуга или для каких-либо иных целей. Такие программы называются *прикладными*.

В прикладных программах могут использоваться команды, слишком сложные для их непосредственного восприятия ПВМ без какой-либо помощи со стороны вспомогательных средств. В этом случае некоторая программа второго типа переводит текст прикладной программы на язык команд, понятных ПВМ. Такой перевод может быть выполнен заблаговременно с помощью *компилятора*, который подготавливает преобразованный вариант прикладной программы для ее более позднего использования. В отличие от компилятора интерпретатор способен оперативно, «на ходу», заново переводить прикладную программу на машинный язык при каждом ее использовании. Один интерпретатор — для языка Бэйсик — постоянно находится в одной из областей ПЗУ персональной ЭВМ; остальные располагаются в динамической памяти совместно с прикладной программой.

Прикладные программы обычно рассчитаны на одновременное существование программ другого типа, обеспечивающих жизненно необходимые связи с такими устройствами, как клавиатура, дисплей и печатающее устройство. Благодаря этому прикладная программа просто «*говорит*»: «Получить символ с клавиатуры» или «Вывести эти данные на экран», *не зная*, как это делается. Постоянно хранимые в ПЗУ программы поддерживают связь с большей частью компонентов системы, а специальная программа, находящаяся в динамической памяти, обеспечивает обмен информацией с дисками.

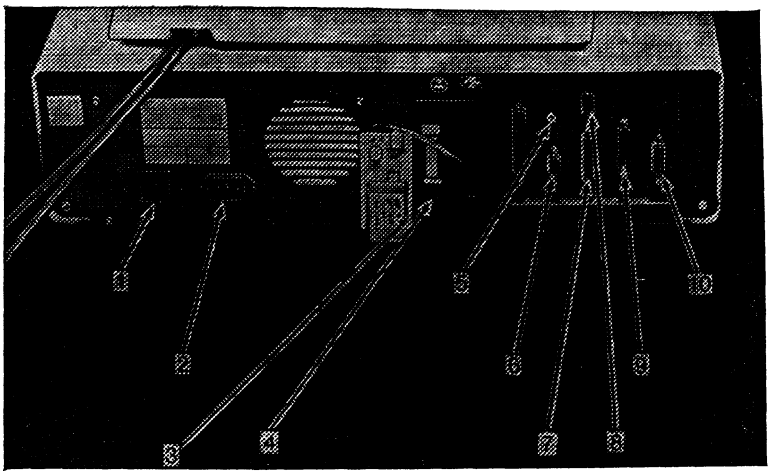


Рис. 1.9. Задняя панель системного блока (возможно другое расположение гнезд).
1 — разъем источника питания монохроматического монитора; *2* — разъем источника питания системного блока; *3* — антенный вход; *4* — разъем клавиатуры; *5* — гнездо для подключения комбинированного монитора; *6* — разъем трехцветного монитора; *7* — разъем параллельного печатающего устройства; *8* — разъем монохроматического монитора; *9* — разъем последовательного печатающего устройства или аналогичных аппаратных средств; *10* — разъем блока управления электронными играми.

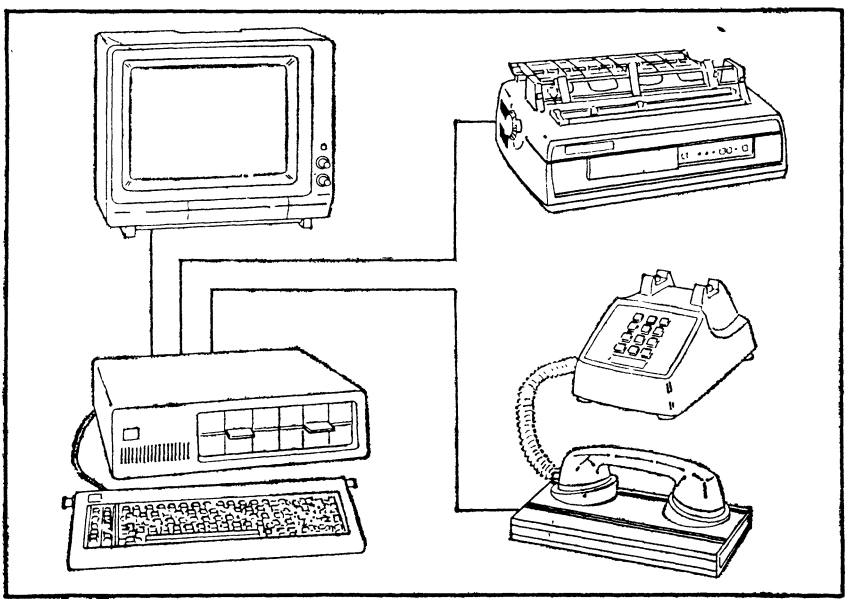


Рис. 1.10. Схема соединения компонентов вычислительной системы.

Стандартная программа, управляющая таким обменом, называется *дисковой операционной системой ПВМ* (ДОС ПВМ), но возможны и другие аналогичные программы.

Монтаж вычислительной системы

Прежде чем пользоваться ПВМ, ее надо правильно собрать и наладить. Для этого все устройства машины снабжены соединительными кабелями, которые подсоединяются либо непосредственно к задней панели системного блока, либо к другому устройству, подключаемому в свою очередь к системному блоку (рис. 1.9). У большинства устройств имеется шнур питания, который включается в штепсельную розетку.

Поскольку ПВМ может иметь различную конфигурацию, рассмотреть все конкретные варианты невозможно. Поэтому до начала работы на ПВМ надо проверить по инструкции каждый ее узел и, убедившись в правильности его коммутации, зарисовать необходимую для использования в будущем скелетную схему соединений компонентов (рис. 1.10).

Глава 2

МОНИТОР, КЛАВИАТУРА И ПЕЧАТАЮЩЕЕ УСТРОЙСТВО

Помимо видеодисплея, клавиатуры и системного блока, которые образуют ядро любой вычислительной системы на основе ПВМ, в большинстве ПВМ имеется еще и печатающее устройство. Данная глава знакомит читателя со способами использования всех перечисленных компонентов вычислительной системы.

Включение и выключение питания

Простейшая ПВМ, содержащая системный блок, монохроматический монитор и клавиатуру, включается и выключается очень легко, поскольку все три ее узла получают питание через один и тот же большой тумблер, окрашенный в красный цвет и расположенный на боковой панели системного блока. С помощью этого тумблера осуществляются также включение и выключение питающего напряжения для дисководов, установленных внутри системного блока.

В случае более сложных вычислительных систем операции включения и выключения оборудования осуществляются бо́льшим числом тумблеров, так как каждый дополнительный узел ПВМ снабжается собственным выключателем. При этом, однако, действует одно общее правило: «Включай внешние узлы ПВМ в любом порядке, но делай это до того, как включишь системный блок». Относительно выключения вычислительной системы справедливо обратное правило: «Сначала выключи системный блок, а затем выключай внешние узлы в любом порядке». Само собой разумеется, что в тех случаях, когда инструкция по эксплуатации конкретного компонента ПВМ устанавливает иную последовательность действий, следует придерживаться рекомендаций этой инструкции.

Работа с видеодисплеем

Жизненно важным элементом ПВМ является экран видеодисплея, хотя пользование им не составляет никакого труда. Если бытовой телевизор в этой роли еще требует некоторых специальных операций, то для работы с видеомонитором не нужно практически никаких навыков.

Следует, однако, иметь в виду ряд общих положений, игнорирование которых может привести к необратимым повреждениям экрана. Прежде всего важно знать, что сохранение на экране изображения без изменений в течение 10 мин приведет к постепенному прожогу электронно-лучевой трубки. Остающийся в результате «отпе-

чаток» изображения может впоследствии навсегда вывести из строя экран дисплея. При интенсивном использовании текстового процессора, программы табличных вычислений или каких-либо других средств, выводящих на экран строки текстовой информации (в отличие от графических данных), в конечном итоге на экране остаются горизонтальные теневые полосы, заметные даже при выключенном состоянии дисплея. Необходимо поэтому соблюдать особую осторожность при использовании в качестве дисплея домашнего телевизора, так как появление на нем упомянутых выше дефектов может отрицательно повлиять на качество приема обычных телепередач.

◆◆◆ ПВМ, укомплектованная и монохроматическим, и цветным мониторами, допускает одновременное использование лишь одного из них. Необходимое переключение может осуществляться программным способом (гл. 15).

Применение в качестве дисплея домашнего телевизора

Для использования бытового телевизора совместно с ПВМ как видеомонитора необходимо прежде всего убедиться, что он в этом случае правильно подсоединен к системному блоку через высокочастотный модулятор (см. рис. 1.2). При этом телевизор должен быть настроен на нужный канал, который определяется инструкцией по использованию модулятора (обычно в ней указывается 33-й канал). Далее антенный выключатель устанавливается в положение, при котором видеосигнал ПВМ подается на вход телевизионного приемника (рис. 2.1), и производится обычное включение телевизора. Однако регулятор громкости при этом должен оставаться полностью выведенным, поскольку ПВМ не использует динамик приемника, а работает лишь с каналом изображения.

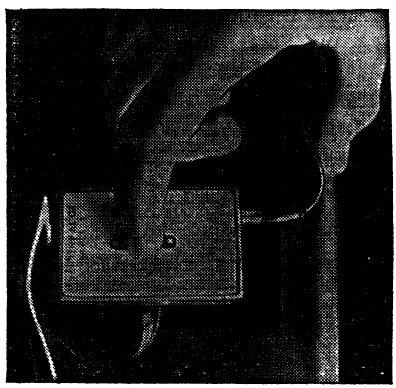


Рис. 2.1. Установка ползункового антенного выключателя в положение, соответствующее использованию домашнего телевизора в качестве дисплея ПВМ.

Если ваш домашний телевизор — цветной, то вначале требуется настроить его на просмотр широковещательных передач с той лишь разницей, что ручки регулирования уровня красного и зеленого цвета должны быть установлены в положение, соответствующее максимальному уровню зеленого. Если с помощью этих органов настройки не удается добиться хорошего качества воспроизведения цветов изображений, выдаваемых ПВМ, следует продолжить подре-

гулировку вплоть до получения приемлемой цветопередачи. При этом, однако, не надо ждать от телевизора высокой четкости изображения: даже телевизионные приемники лучших марок дают «картинку», которая выглядит размытой в сравнении с тем же изображением на видеомониторе ПВМ.

Необходимо иметь в виду, что домашние телевизоры не воспроизводят принимаемое изображение целиком. В них неизбежно происходит увеличение принимаемого кадра, в результате чего часть его, расположенная по краям трубки, теряется. Но зато это препятствует появлению на экране неприглядной черной окантовки изображения. Такая защитная мера называется *разверткой за пределами экрана*; в одних телевизорах она выражена более явно, чем в других, но даже в случае совсем незначительного ее использования наличие развертки за пределами экрана может служить серьезным препятствием для работы ПВМ со всей его площадью при выводе изображений. Любые такие попытки будут неизбежно приводить к потере части изображения, подобно тому как это происходит при показе слайдов или кинофильма на экране слишком малых размеров. Но хотя ПВМ и не способна сжимать выводимые данные в целях устранения эффекта развертки за пределами экрана, она может сдвигать изображение на экране влево или вправо. Один из способов реализации подобного сдвига объясняется в гл. 4.

Что же касается видеомонитора, то он более прост в обращении, чем домашний телевизор, так как не требует ни выбора специального канала, ни установки в нужное положение антенного выключателя. Обычно бывает достаточно настроить органы управления экраном один раз и больше не изменять их положения.

Информация, видимая на экране дисплея

ПВМ начинает выводить данные на экран через несколько секунд после включения машины и дисплея. Первое, что появляется на экране, это мерцающая подчеркивающая черточка в его левом верхнем углу. (В телевизорах с избыточной разверткой за пределами экрана мерцающая черточка при отсутствии информации может оказаться по этой причине невидимой.) Через короткое время после этого высвечивается сообщение, показанное на рис. 2.2.

Курсор. Мерцающая подчеркивающая черточка, появляющаяся на экране сразу после включения вычислительной системы, называется *курсором*; она указывает позицию расположения на экране очередного символа. Курсор может принимать и иную форму, в том числе форму прямоугольника или квадрата, и даже может быть невидимым для глаза. В последнем случае местоположение курсора остается известным машине, хотя и не обнаруживается визуально.

Изменение формы курсора и его видимости осуществляется программным способом и рассматривается в гл. 11.

Самоконтроль ПВМ. В первые несколько секунд после включения системного блока ПВМ осуществляет контроль собственных узлов.

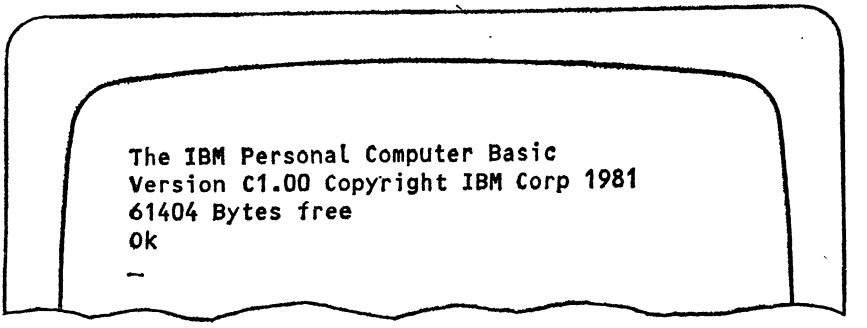


Рис. 2.2. Типичный кадр, появляющийся на экране дисплея при включении ПВМ: «Бэйсик ПВМ фирмы IBM. Версия C1.00. Авторское право фирмы IBM, 1981. Свободная память 61404 байт».

При обнаружении неполадок она выдает на экран то или иное кодовое сообщение; смысл всех таких сообщений детально разъясняется в разд. 4 «*Руководства по эксплуатации ПВМ фирмы* IBM».

Работа на клавиатуре

Клавиатура ПВМ в значительной степени сходна со стандартной клавиатурой обычной пишущей машинки. Клавиши всех букв алфавита, цифр от 0 до 9 и большинства знаков пунктуации расположены точно в тех местах, которые привычны для человека, печатающего на машинке. Однако имеется и целый ряд особенностей в связи с тем, что клавиатура ПВМ насчитывает 83 клавиши, объединенные в 4 группы (рис. 2.3).

Клавиши, относящиеся к одной группе, располагаются вместе и имеют одинаковый цвет. Центральное положение занимает самая многочисленная группа светлых клавиш стандартной клавиатуры пишущей машинки. По обе стороны от нее размещаются вертикально стандартные управляющие клавиши темного цвета; многие из них также имеются в пишущих машинках. В крайней правой группе скомпонованы светлые клавиши, которые служат двум целям: во-первых, они могут выполнять роль цифровой клавиатуры калькулятора и, во-вторых, с их помощью можно управлять положением курсора на экране. Крайнюю левую группу образуют темные клавиши, называемые *функциональными* или *программируемыми*; их назначение может меняться программным способом.

Работа с буквенными и цифровыми клавишами не должна вызывать никаких затруднений, если для пользователя привычна клавиатура пишущей машинки. Правда, некоторые знаки пунктуации могут оказаться на клавиатуре ПВМ в иных местах, однако к новому

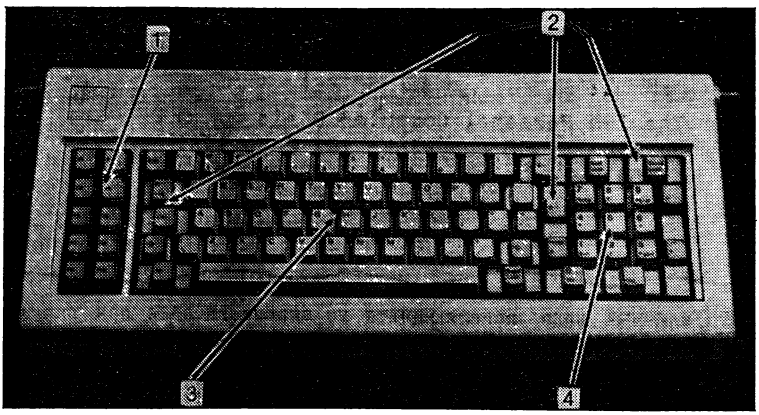


Рис. 2.3. Группы клавиш на клавиатуре ПВМ.
1 — функциональные (программируемые) клавиши; *2*—стандартные управляющие клавиши; *3* — клавиши пишущей машинки; *4* — малая цифровая клавиатура для набора цифр или управления курсором.

их расположению можно быстро привыкнуть. Три другие группы клавиш нуждаются в более детальном рассмотрении и описываются ниже.

Стандартные управляющие клавиши

Это клавиши темного цвета, расположенные слева и справа от группы клавиш пишущей машинки. Они реализуют стандартные функции управления; часть из них может блокироваться или переопределяться в прикладных программах; некоторые служат для проверки результатов выполнения рабочих команд. Рассмотрим функциональное назначение каждой из управляющих клавиш.

Esc — сокращенное написание слова «Escape» («Переход»). Клавиша с таким обозначением вызывает игнорирование машиной любых символов, записанных в текущей строке экрана, которая в некоторых случаях при этом стирается. В ряде других ситуаций при нажатии клавиши **Esc** в конце текущей строки появляется символ косой черты \, и курсор перебрасывается в начальную позицию следующей строки.

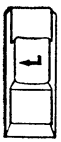
На нижнем регистре эта клавиша, обозначаемая символом →, действует подобно клавише табуляции в пишущей машинке, перемещая курсор на 8 позиций вперед, в направлении очередной метки табуляции. На верхнем регистре она не оказывает никакого воздействия.

Наименование этой клавиши **Ctrl** представляет собой сокращенное написание слова «Control» («Управление»). Клавиша с таким обозначением действует в значительной мере подобно клавише смены регистра в пишущей машинке — в том смысле что она, будучи нажатой совместно с какой-либо другой клавишей, изменяет ее действие. Некоторые наиболее часто встречающиеся комбинации этой клавиши с другими будут рассмотрены позже в этой же главе.

Для обеспечения бóльшего удобства имеются две клавиши ⇧, любая из которых может использоваться аналогично клавише смены регистра в пишущей машинке при необходимости перехода на заглавные (прописные) буквы или набора символов, обозначенных в верхней части цифровых и пунктуационных клавиш. Если клавиатура настроена на работу с заглавными буквами и заперта в этом режиме (см. ниже пояснение к клавише **Caps Lock**), клавиши ⇧ действуют как ключи *нижнего регистра*, обеспечивающие в нажатом состоянии набор строчных букв; при этом, однако, они не выводят клавиатуру из режима прописных букв.

Alt — сокращение от «Alternate» («Изменение»). Подобно клавишам ⇧ и **Ctrl**, клавиша **Alt**, будучи нажатой совместно с некоторой другой, модифицирует действие последней.

Действие этой клавиши, условно обозначаемой символом ←, аналогично действию клавиши возврата на одну позицию каретки пишущей машинки. Разница состоит лишь в том, что в данном случае происходит не только возврат курсора на одну позицию, но и стирание находящегося в ней символа.



Действие клавиши ↵ более всего сходно с действием клавиши возврата каретки с переводом строки, которую можно найти на клавиатуре многих электрических пишущих машинок. В документации некоторых ЭВМ клавиша ↵ называется *клавишей возврата* либо *клавишей ввода*, так как ее назначение состоит в том, чтобы завершить набор текущей строки и перейти к следующей.

PrtSc — сокращенное написание слов «Print Screen» («Распечатка кадра»). При индивидуальном нажатии клавиши **PrtSc** на экране появляется «звездочка»; если же одновременно оказывается нажатой клавиша ⇧, то информация, видимая на экране, воспроизводится печатающим устройством. Для это-

го, разумеется, необходимо иметь такое устройство в комплекте ПВМ и обеспечить его готовность к работе. (Операции вывода информации на печать будут рассмотрены в конце данной главы.)

Клавиша **Caps Lock** запирает клавиатуру в режиме заглавных букв, благодаря чему для набора их не требуется одновременного нажатия клавиши ⇧. В действительности при нажатии клавиши ⇧ в этом режиме одновременное использование любой буквенной клавиши приводит к появлению на экране букв нижнего регистра. Действие клавиши **Caps Lock** отличается от эффекта клавиши блокировки регистра в пишущей машинке тем, что **Caps Lock** охватывает лишь 26 букв английского алфавита. Например, для набора вопросительного знака необходимо всегда нажимать совместно клавиши ⇧ и /. Режим заглавных букв устанавливается путем нажатия клавиши **Caps Lock**; при повторном ее нажатии происходит возврат к строчным буквам. Ни по каким внешним признакам невозможно определить, включен или отключен режим заглавных букв. (В гл. 15 будет описана короткая программа, позволяющая получить ответ на этот вопрос.)

Сама по себе клавиша **Scroll Lock** («Блокировка прокрутки») не приводит ни к каким действиям, однако в большинстве случаев при одновременном нажатии еще и клавиши **Ctrl** машина останавливается независимо от выполняемых в это время операций и ожидает ввода команды с клавиатуры.

Двухрежимная малая цифровая клавиатура

Группа светлых клавиш в правой части клавиатуры по своей компоновке очень напоминает клавишную панель стандартного конторского калькулятора. Это облегчает ввод чисел в машину, особенно для тех пользователей, которые умеют работать на клавиатуре калькулятора слепым методом. Расположенные по соседству клавиши **PrtSc**, + и — несут на себе символы умножения, деления и вычитания, что еще больше усиливает сходство рассматриваемой группы светлых клавиш с клавишной панелью калькулятора.

Клавиша **Num Lock** («Цифровая блокировка») определяет выбранный режим работы малой клавиатуры: цифровой или нецифровой. Переключение режима осуществляется с каждым нажатием этой клавиши. Выявить действующий режим по каким-либо внешним признакам невозможно: единственный выход состоит в том, чтобы нажать одну из клавиш и посмотреть, что при этом произойдет. (В гл. 15 будет приведена небольшая программа, позволяющая получать информацию о текущем режиме работы малой цифровой клавиатуры.)

В цифровом режиме каждая из светлых клавиш рассматриваемой

группы генерирует одну из цифр от 0 до 9 или десятичную точку соответственно нанесенному на ней обозначению.

В нецифровом режиме цифровая клавиатура обычно используется для управления положением курсора на экране дисплея, но не всегда. Например, в дисковой операционной системе ПВМ (ДОС ПВМ) нецифровой режим интерпретируется по-своему (подробнее об этом см. гл. 5).

Клавиша **Home** («Исходное положение») служит для перемещения курсора в левый верхний угол экрана, в *исходную позицию.*

Клавиша **End** («Конец») обеспечивает перемещение курсора в конец текущей строки.

С помощью клавиши **Ins** («Вставка») машина переводится в режим вставки и выводится из него. В этом режиме курсор принимает форму не черточки, а квадрата, и каждый набираемый символ появляется точно на месте курсора, как бы сдвигающего всю остальную часть строки вправо. При этом вставляемый символ не замещает указываемый курсором знак, как в случае, когда первый имеет форму черточки.

Посредством клавиши **Del** («Вычеркивание») производится удаление с экрана того символа, на который указывает курсор. Все символы, расположенные справа от исключаемого, сдвигаются при этом на одну позицию влево в целях заполнения освободившегося места.

Клавиша ↑ обеспечивает перемещение курсора на одну строку вверх. Если же курсор уже стоит в самой верхней строке экрана, нажатие этой клавиши не дает никакого эффекта.

Клавиша ↓ обеспечивает перемещение курсора на одну строку вниз. Если же курсор уже стоит в самой нижней (24-й) строке экрана, то нажатие этой клавиши ничего не изменяет.

Клавиша ← обеспечивает перемещение курсора на одну позицию влево. Если же курсор уже стоит в левом краю экрана, то с нажатием этой клавиши он передвигается в крайнюю правую позицию предыдущей строки. В том случае, когда курсор занимает исходное положение, нажатие клавиши ← не вызывает никаких изменений.

Клавиша → обеспечивает перемещение курсора на одну позицию вправо. Если же курсор уже стоит в правом краю экрана, то с нажатием этой клавиши он передвигается в крайнюю левую позицию следующей строки. В том случае, когда курсор занимает положение в крайней правой позиции самой нижней строки, при нажатии клавиши → происходит вставка пустой строки в нижней части экрана. Для высвобождения места под эту пустую строку с экрана удаляется самая верхняя строка.

Функциональные клавиши

Два столбца клавиш темного цвета в левом краю клавиатуры могут программироваться на формирование различных цепочек символов. В некоторых прикладных программах эти клавиши могут использоваться особым образом, о чем говорится в прилагаемых к программам инструкциях по их эксплуатации. В гл. 11 будет показано, как можно осуществить программирование функциональных клавиш по своему усмотрению.

Комбинации клавиш

Как отмечалось выше, целый ряд клавиш на клавиатуре ПВМ не имеет собственного самостоятельного функционального назначения; такие клавиши лишь изменяют результат действия других. Примерами могут служить клавиши ⇧, **Ctrl** и **Alt**. При использовании одной из этих клавиш ее необходимо удерживать в нажатом состоянии и одновременно нажимать другую. Всюду далее подобные совместные нажатия клавиш обозначаются с помощью разделяющей вертикальной черты. Так, например, запись **Ctrl|Scroll Lock** означает совместное нажатие клавиш **Ctrl** и **Scroll Lock**. Существует довольно много допустимых комбинаций клавиш, реализуемых при программировании; особенно часто используется в различных комбинациях клавиша **Alt**.

При наборе комбинации **Ctrl|Alt|Del** с машиной происходит примерно то же, что и при включении и выключении питания. Однако в случае возвращения ПВМ в исходное состояние с помощью именно этой комбинации клавиш не происходит полного самоконтроля ее узлов.

Посредством комбинации **Ctrl|Num Lock** можно зафиксировать изображение на экране дисплея. При этом экран переводится в режим приостановки активности: ничего нового на нем не может появиться до тех пор, пока он не будет разблокирован нажатием любой клавиши, кроме **Ctrl**, ⇧, **Alt**, **Caps Lock**, **Num Lock** или **Scroll Lock**. Нажатие клавиш в комбинации **Ctrl|Num Lock** возможно в любой момент работы на ПВМ.

Комбинация **Ctrl|Scroll Lock** вызывает прерывание работы машины и перевод ее в режим ожидания ввода команды с клавиатуры. Этот процесс называется *приостановкой* (*break*), вследствие чего на передней грани клавиши **Scroll Lock** и выгравировано обозначение Break.

Автоматическое повторение набора символов

Читатель, вероятно, уже имел возможность убедиться на практике, что при удержании некоторой клавиши в нажатом состоянии происходит автоматическое повторение набора соответствующего

ей символа. Такое повторное действие характерно для всех клавиш, исключая **Ctrl**, ⌂, **Alt**, **Ins** и **Num Lock** при их индивидуальном использовании. Однако при нажатии клавиш **Ctrl**, ⌂ или **Alt** совместно с другими эффект повторения набора сохраняется.

Ввод команд с клавиатуры

В тех случаях, когда отсутствуют какие-либо иные признаки активного поведения системы, кроме мерцания курсора, ПВМ скорее всего находится в режиме ожидания ввода команды. Эта операция осуществляется путем нажатия определенной последовательности клавиш. Машина, управляемая программой, анализирует произведенный набор и пытается определить требуемую последовательность действий. При правильном наборе всей цепочки символов управляющая программа обеспечит выполнение машиной введенной вами команды. Таким образом, программа (а не сама машина) определяет правильность набора команды.

В каждой программе используется вполне определенная совокупность команд, и если такая программа осуществляет управление ПВМ, то это будет единственный набор команд, «понимаемых» машиной. Функции управления может выполнять, например, некоторая прикладная программа — скажем, программа обработки текстовой информации. В какие-то моменты управление машиной может осуществляться дисковой операционной системой или интерпретатором Бэйсика. Все это означает, что при вводе команды с клавиатуры нужно четко знать, какая программа играет роль управляющей; по мере приобретения навыков работы на ПВМ это удается легко определять по контексту, видимому на экране дисплея. Как только получен ответ на вопрос о том, какой программе передано управление, у пользователя есть возможность обратиться к инструкции по ее использованию или к руководству по эксплуатации для выяснения перечня команд, воспринимаемых данной программой. (В гл. 3 и 5 описывается система команд ДОС ПВМ, а в гл. с 6-й по 15-ю команды языка Бэйсик.)

Команда не обязательно является принадлежащей исключительно какой-то одной программе. Однако следует все же проявлять определенную осторожность, поскольку сходные по написанию команды могут сильно отличаться по результатам их выполнения в зависимости от конкретной решаемой задачи. Иначе говоря, некоторая последовательность нажатий клавиш может означать нечто определенное для одной программы, то же самое для какой-то другой программы, иметь совершенно иной смысл для третьей и быть совершенно бессмысленной для четвертой.

В связи с тем что команды имеют неодинаковую длину, редко практикуется ограничение числа набираемых на клавиатуре символов. Управляющая программа не реагирует на команду до тех пор,

пока ей не будет сообщено об окончании набора команды. Почти универсальным признаком конца набора является нажатие клавиши с обозначением ↵.

♦♦♦ Если вы набрали команду и долго ожидаете, пока что-нибудь произойдет, то знайте, что вы, вероятно, забыли нажать клавишу ↵. Существуют команды, которые вообще не нуждаются в признаке конца набора, но есть и такие, которые требуют иного признака конца, отличного от рассмотренного выше. И в том и в другом случае конкретные условия оговариваются в инструкции по использованию конкретной программы.

Исправление ошибок клавишного набора

Если при работе с клавиатурой вы допустите ошибку, то машина либо выполнит неверные действия, либо вообще не проявит никакой реакции. Очень трудно найти человека, который никогда бы не ошибался, поэтому в ПВМ предусматриваются средства исправления ошибок. Ошибки, обнаруженные до нажатия клавиши ↵, устраняются с помощью клавиши возврата на одну позицию, которая нажимается до тех пор, пока не будут стерты все неверно набранные символы. После этого набор повторяется более тщательно. В некоторых случаях бывает удобнее стереть сразу всю строку и начать ее повторное заполнение, что обеспечивается нажатием клавиши Esc. Существует и множество других способов исправления ошибок, но они меняются от программы к программе. Подробную информацию о них можно получить в результате детального ознакомления с инструкциями по использованию соответствующих программ. В гл. 5 будет показано, как должны исправляться ошибки при работе в операционной среде ДОС ПВМ, а в гл. 6 будут рассмотрены процедуры редактирования команд Бэйсика.

Иногда может возникнуть ситуация, при которой скорость ввода данных будет превосходить пропускную способность ПВМ. В подобных случаях машина подает предупредительный звуковой сигнал через встроенный динамик. В ответ на этот сигнал необходимо прекратить работу на клавиатуре на несколько секунд, а затем осторожно попытаться ввести очередной символ. Отсутствие звукового сигнала будет означать возможность продолжения работы с клавиатурой. Если же машина все еще будет перегружена и не сможет принимать данные с клавиатуры, она повторно подаст звуковой сигнал.

Работа с печатающим устройством

Процедуры работы с печатающими устройствами не отличаются большим разнообразием, несмотря на широкий спектр функциональных возможностей устройств различных типов. Имеющееся

разнообразие касается скорее производительности печати, а не самих процедур. Одни печатающие устройства работают быстро, другие — медленно; разными оказываются четкость печати и размеры символов, а также ряд других характеристик, однако управление этими функциональными возможностями обычно является прерогативой программ. В данной главе рассматриваются лишь рабочие процедуры, применимые ко всем печатающим устройствам. Конкретные подробности, касающиеся печатающего устройства, которое имеется в комплекте вашей ПВМ, следует уточнять, обратившись к инструкции по эксплуатации этого устройства.

Совместимость ПВМ с печатающими устройствами

Как отмечалось в гл. 1, и ПВМ, и печатающее устройство должны работать в одинаковом режиме передачи данных — параллельном или последовательном. На задней панели системного блока имеются отдельные гнезда для того и другого режима, однако в связи с тем, что машина не способна самостоятельно определять, к какому из двух гнезд подключено печатающее устройство, она по принципу умолчания устанавливает параллельный режим передачи. При каждом включении ПВМ необходимо изменять это условие, если предполагается использовать печатающее устройство последовательного действия. В ДОС ПВМ имеется специальная команда MODE (РЕЖИМ), с помощью которой вычислительная система настраивается на работу с последовательным устройством печати вместо параллельного (гл. 3). Некоторые ПВМ фирмы IBM, отправленные заказчикам до июня 1982 г., могут быть снабжены операционной системой ДОС ПВМ старой версии 1.0, не предусматривающей команды MODE для печатающих устройств. Такие ПВМ должны быть просто переоснащены новой версией ДОС ПВМ.

При последовательном режиме связи ПВМ способна пересылать информацию на устройство печати с любой из 15 скоростей. Эти скорости измеряются в *бодах* и иногда выражаются в *битах в секунду* (бит/с) [1]. Аналогичные скорости печатающих устройств обычно тоже устанавливаются по выбору. Для обеспечения эффективной работы устройства печати его скорость в бодах должна соответствовать скорости, с которой ПВМ выдает информацию на печать. Что касается ПВМ, то ее скорость передачи данных регулируется программным способом; в печатающих устройствах она обычно задается внутренними схемами коммутации. Стандартная для ПВМ скорость передачи составляет 2400 бод, однако эту скорость можно изменить с помощью команды MODE, описанной в гл. 3, либо она может изменяться используемой прикладной программой. В случае неясности этого вопроса необходимо обратиться за уточнениями к

[1] Каждый символ отображается восемью битами.

инструкции по использованию конкретной программы или к ее поставщику.

Гораздо более сложные проблемы обеспечения совместимости возникают при использовании таких программно-управляемых функциональных возможностей, как печать нижних индексов, выравнивание строк по правой границе, подчеркивание и выбор типа шрифта. Программы осуществляют управление подобными операциями путем посылки на печатающее устройство определенных последовательностей непечатаемых символов. К сожалению, в различных моделях печатающих устройств для реализации одинаковых возможностей могут использоваться совершенно разные управляющие последовательности символов. Здесь спасает то, что в большинстве программ, ориентированных на использование расширенных возможностей печати, задаются управляющие последовательности для нескольких типов печатающих устройств, так что пользователь может выбирать конкретный вариант по своему усмотрению. Но для этого необходимо учесть рекомендации по выбору устройства печати, содержащиеся в инструкции по использованию конкретной программы.

Органы управления печатающим устройством

Типичное печатающее устройство бывает всегда снабжено выключателем питания, несколькими кнопками ручного управления и целым рядом лампочек индикации состояния на верхней или лицевой стороне управляющей панели (рис. 2.4). Иногда силовой

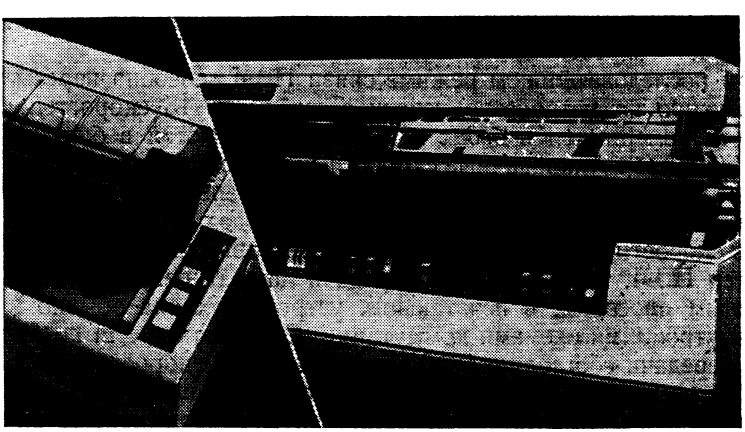


Рис. 2.4. Органы ручного управления на двух печатающих устройствах разных типов.

выключатель располагается на боковой или задней стороне печатающего устройства; все прочие управляющие переключатели и рычаги могут размещаться внутри его. В табл. 2.1 перечислены наи-

Таблица 2.1. **Стандартные органы управления и индикаторы состояния печатающих устройств**

Наименование [1])	Функциональное назначение и сигнализируемое состояние
	Переключатели и кнопки
Power	Запуск и останов печатающего устройства
Online (Select)	Приостановка/продолжение печати
Form Feed (FF)	Прогон бумаги до следующей страницы
Line Feed (LF, Paper adv.)	Протяжка бумаги на одну строку
Clear (Reset)	Установка устройства в исходное положение после устранения ошибки
Override	Уведомление об истощении запаса бумаги; прекращение печати текущей страницы
	Лампочки индикации состояния (при подсветке)
Power	Питание включено
Ready	Устройство готово к работе; блокировочные и защитные ключи проверены
Select (Online)	Разрешено продолжение печати
Paper Out (Paper)	Нет бумаги
Alarm (Error)	Конец красящей ленты или внутренняя неисправность

[1]) В некоторых устройствах используются иные наименования.

более часто встречающиеся в печатающих устройствах органы управления и лампочки индикации состояния.

В большинстве печатающих устройств происходит автоматическое распознавание числа строк, остающихся до конца страницы, и выбрасывание напечатанного листа при нажатии соответствующей кнопки. В случае использования рулонной бумаги или пачки отрывных листов наличие такой возможности обеспечивает прогон бумаги на начало очередной страницы, однако для правильного автоматического формирования страниц необходима тщательная настройка первой страницы перед запуском устройства печати, с тем чтобы обеспечить его механизму начальную точку отсчета строк. В некоторых печатающих устройствах предусматривается специальная кнопка, при нажатии которой в качестве начала страницы задается текущая строка.

В целом ряде печатающих устройств имеется переключатель для установки автоматического или локального режима перевода стро-

ки. Этот переключатель может располагаться на лицевой панели или под крышкой устройства печати; его нормальное положение — выключенное.

Подготовка печатающего устройства к работе

При подготовке устройства печати к работе необходимо первым делом выбрать нужный тип бумаги: для непрерывной распечатки, для многоэкземплярной печати, для оформления фирменных документов, отправляемых почтой, и т. п. При постраничном печатании документов возможно использование режима одноэкземплярной печати, если это допускает имеющееся устройство. Выбрав тип бумаги, устанавливают ее в печатающее устройство и выравнивают по вертикали и горизонтали.

Закончив установку бумаги, следует проверить, не изношена ли красящая лента, и отрегулировать ее должным образом или поставить новую. Если в печатающем устройстве предусмотрены сменные печатающие узлы (*лепестковый шифроноситель* или *цилиндрическая головка*), то надо убедиться в том, что выбран именно нужный элемент и что он установлен с соблюдением всех правил. Чтобы вычислительная машина могла адаптироваться к используемому типу бумаги, надо переключить в требуемое положение все указатели плотности печати или толщины бумаги печатных форм.

В заключение закрывают все крышки печатающего устройства: при открытых крышках срабатывают блокировочные переключатели, которые временно отключают устройство печати. Затем включают питание и устанавливают в рабочее положение ключ «Select» («Выборочная печать») или «Online» («Оперативный режим»), если таковой имеется. О готовности устройства к работе будут сигнализировать лампочки «Ready» («Готово») и «Select» («Выборочная печать»).

Глава 3

ДИСКИ И ДИСКОВАЯ ОПЕРАЦИОННАЯ СИСТЕМА ДОС ПВМ

Наиболее важным узлом ПВМ является дисковод, так как при наличии последнего машина может выполнять более широкий круг задач, в том числе большой размерности и сложности; самая распространенная разновидность накопителей для ПВМ — дискеты. В данной главе описываются методы выбора дискетов и способы их использования, а также дисковая операционная система и ее команды, полный перечень которых дается в приложении В.

Работа с дискетами

Чтобы обеспечить долговечность дискета, достаточно хрупкого изделия, необходимо обращаться с ним крайне осторожно. Важное значение имеет также правильный отбор пустых дискетов, так как внешне они практически неразличимы.

Выбор нужных дискетов

Несмотря на стандартную конструкцию дискетов (рис. 3.1), некоторые их свойства имеют ряд особенностей, обусловленных различием характеристик дисководов. Эти особенности могут быть как очевидными (и тогда их легко обнаружить), так и неявными. В левом верхнем углу пустых дискетов обычно имеется постоянная наклейка с указанием характеристик или по крайней мере номера модели, по которому эти характеристики могут быть определены.

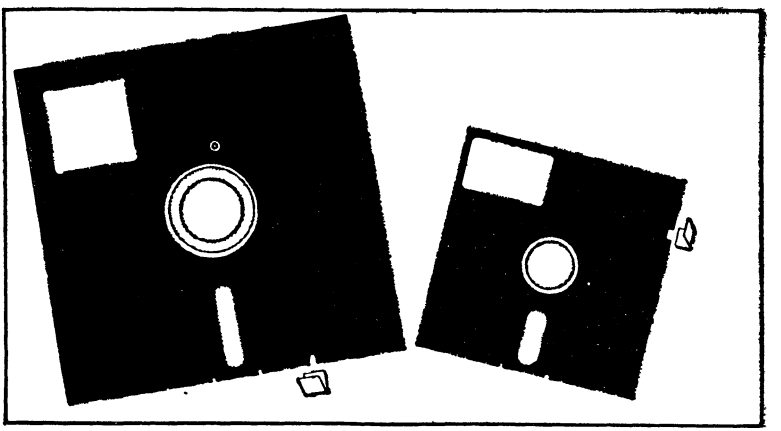


Рис. 3.1. Детали дискетов диаметром 8 дюймов (слева) и $5^1/_4$ дюйма (справа).

Неправильно подобранный дискет может выйти из строя сразу или проработав какое-то время и таким образом свести на нет многие часы или дни кропотливой работы.

В дискетных накопителях фирмы IBM, которыми комплектуются ПВМ, выпускаемые этой фирмой, применяются программно-секционированные дискеты диаметром $5^1/_4$ дюйма с удвоенной плотностью записи. В подобных накопителях применяются как односторонние, так и двусторонние дискеты, причем в последних обе стороны используются лишь в накопителях емкостью 320К.

◆◆◆ Никогда не следует в ПВМ с дискетными накопителями фирмы IBM пользоваться дискетами с обычной плотностью записи: рано или поздно возникнет ситуация, при которой неизбежна потеря информации.

Уход за дискетами

Дискеты — прецизионные устройства и не терпят плохого обращения: при постоянной неправильной эксплуатации они выходят из строя (не говоря уже о том, что такая эксплуатация может явиться причиной повреждения самого дисковода). Утрачиваемая при этом информация может быть иногда частично восстановлена; полного восстановления удается добиться крайне редко.

В дискетах не предусмотрена идеальная защита записанных на них данных: защитные конверты рассчитаны только на предохранение поверхности носителей от повреждений на коротком пути от дискотеки до дисковода. Поэтому необходимо строго следовать указаниям, напечатанным на тыльной стороне защитных конвертов. Особую осторожность надо соблюдать при нанесении надписей на наклейке, прикрепленной к дискету, так как даже давление карандаша или шариковой ручки может оказаться достаточным, чтобы через защитный конверт повредить магнитный слой диска. Во избежание порчи информации, записанной на дискетах, последние следует хранить подальше от телевизоров, видеомониторов, звонящих телефонов и других источников магнитных полей.

Маркировка

Как уже отмечалось, внешне дискеты практически неразличимы. Поэтому для их идентификации удобнее использовать съемные наклейки, а не постоянные. Учитывая, что наличие любой неоднозначной метки лишь немногим лучше, чем отсутствие всякой метки, необходимо строго следить за индивидуальностью меток. Хорошо также иметь полный перечень содержимого дискета.

Вставление дискета в дисковод

Чтобы вставить дискет в дисковод, крышка последнего должна быть открыта. В некоторых дисководах достаточно нажать планку или кнопку, и крышка открывается под действием пружины, однако в большинстве случаев крышку приходиться приподнимать вручную. Если в дисководе уже находится какой-то дискет, его надо

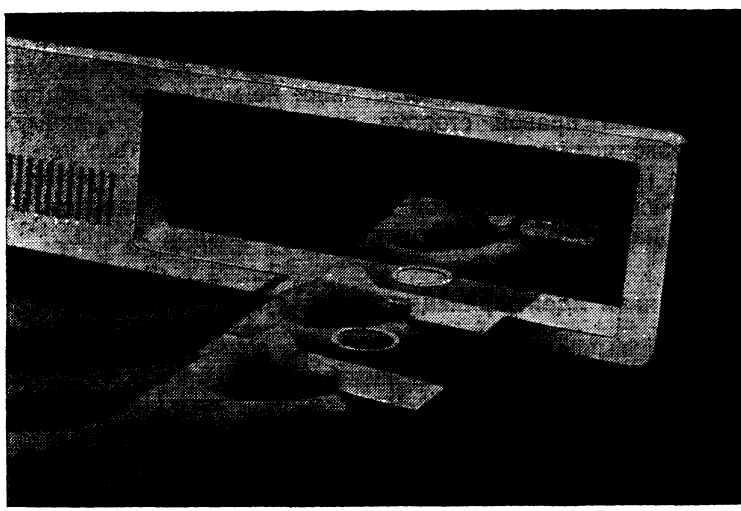


Рис. 3.2. Вставление дискета.

вытянуть на себя, захватив большим и указательным пальцами, стараясь не повредить.

Дискет плавно вводят в дисковод меткой вверх, стараясь не погнуть (рис. 3.2), и осторожно проталкивают внутрь. При ощущении сопротивления движению прекращают проталкивание дискета, вынимают его и, устранив мешающий фактор, повторяют попытку. Если дискет все-таки не вставляется на место, берут другой. В тех случаях, когда ни один дискет не удается вставить в дисковод, это означает, что неисправен сам дисковод и его следует отдать в ремонт.

Затем медленно закрывают крышку дисковода. Если она опустится только на несколько миллиметров, это значит, что дискет вставлен не до конца.

Дискеты с защитой от несанкционированной записи информации

Случайное стирание или повторную запись информации можно исключить, сделав специальный вырез в определенном месте дискета. На $5^1/_4$- и 8-дюймовых дискетах вырез находится в разных местах и соответственно по-разному интерпретируется:

— вырез на правой стороне нижней кромки 8-дюймового дискета защищает дискет от записи, когда он находится в дисководе (рис. 3.1). Поэтому такой вырез называется *прорезью блокировки записи*. Защиту можно снять, закрыв прорезь специальной наклейкой или кусочком непрозрачной ленты. Некоторые 8-дюймовые дискеты не имеют выреза и, следовательно, не могут быть защищены;

— вырез на правой стороне верхней кромки $5^1/_4$-дюймового дискета позволяет производить запись на дискет при его установке в дисковод (рис. 3.1). Этот вырез называется *прорезью разрешения записи*. Чтобы защитить такой дискет, достаточно закрыть вырез маленькой наклейкой или кусочком непрозрачной ленты.

Множественная идентификация дисководов

В составе вычислительной системы на основе ПВМ может быть всего лишь один накопитель на дискетах, но скорее всего таких накопителей будет два или более. В системном блоке ПВМ предусмотрено место для установки интерфейсных плат одного-двух $5^1/_4$-дюймовых дискетов либо одного накопителя такого типа и одного винчестерского диска; для установки большего числа дисководов требуются внешние подключения. В системе с несколькими дисководами каждый из них обозначается одной буквой *A*, *B*, *C* и т. д. для отличия их друг от друга.

Чтобы облегчить управление огромной памятью накопителя на винчестерских дисках, ее часто подразделяют на несколько *томов* или *разделов*, каждый из которых имеет свою собственную метку наподобие дисковода. При этом машина не делает различия между разделом накопителя на винчестерских дисках и накопителем на дискетах.

По причинам, указанным ниже, один дисковод (A) имеет специальное назначение. В системах с двумя встроенными накопителями левый накопитель обычно обозначается как дисковод A. В системах с одним накопителем на винчестерских дисках роль дисковода A может выполнять один из разделов памяти.

Дисковая операционная система

Обращение к какому-либо дисководу является непростым процессом, выполнение которого требует сложного программирования. Все необходимые операции с дисками координируются *дисковой*

операционной системой (ДОС), представляющей собой специальную системную программу. По характеру запросов, которые обслуживает программа дисковой операционной системы, ее действия напоминают работу клерка, ведающего архивом папок с деловыми бумагами, который получает запросы типа «Достаньте мне личное дело Смитерса» или «Поставьте обратно дело ABIG Holding Corporation», или «Продиктуйте мне очередную фамилию и адрес из нашего реестра рассылки».

Для ПВМ создано множество различных дисковых операционных систем, причем все они делают одно и то же, но несколько различными способами. Поскольку большинство прикладных программ хранится на дисках и запускается по командам ДОС ПВМ, пользователь должен иметь представление об этой операционной системе. Ряд других ее команд обеспечивает поддержание в должном порядке информации, записанной на дисках.

Версии ДОС ПВМ

До июня 1982 г. фирма IBM поставляла ДОС ПВМ версии 1.0 («Один, точка, ноль»). Однако эта версия не была рассчитана на применение двусторонних дискетов и не обеспечивала реальных возможностей использования печатающего устройства последовательного действия. В связи с этим в мае 1982 г. фирма IBM приступила к созданию новой версии 1.1, рассчитанной на применение в накопителях одно- и двусторонних дискетов и эффективное использование последовательного печатающего устройства. Поскольку в большинстве случаев затраты на замену версии 1.0 версией 1.1 невелики, ограничимся рассмотрением только ДОС ПВМ версии 1.1.

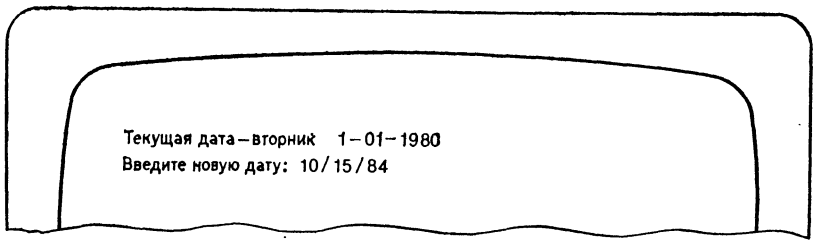
Запуск ДОС ПВМ

Для нормальной работы ДОС ПВМ все дисководы должны находиться во включенном состоянии. Что же касается всех внешних накопителей, имеющих собственные силовые выключатели, то они должны быть включены до того как будет включен системный блок. Встроенные дисководы не имеют отдельных тумблеров подачи питания: оно подводится к ним от системного блока. Если инструкции не содержат никаких иных указаний, то при включении или выключении питания крышки дисководов лучше оставлять открытыми.

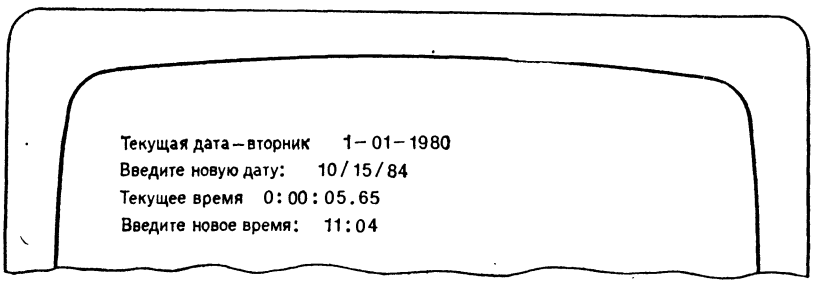
Программа дисковой операционной системы обычно располагается на диске, а не в постоянном запоминающем устройстве ПВМ, и вычислительная система должна каким-то образом пересылать программу ДОС ПВМ с диска в динамическую память машины. Возникает ситуация, при которой ПВМ должна «сама себя поднять за волосы». Эту функцию выполняет маленькая *программа самоза-*

грузки, всегда находящаяся в постоянном запоминающем устройстве ПВМ: она обращается к дисководу А для пересылки дисковой операционной системы в оперативную память ПВМ, т. е. реализует процесс, называемый *загрузкой* или *самозагрузкой* ДОС.

Программа самозагрузки загружает ДОС ПВМ сразу при включении системного блока или сбросе системы в исходное состояние



А Ввод даты

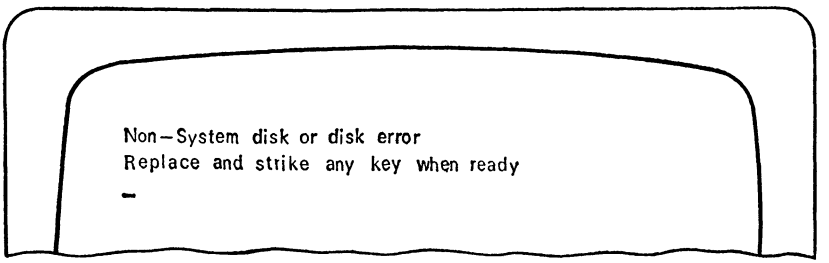


В Ввод времени суток

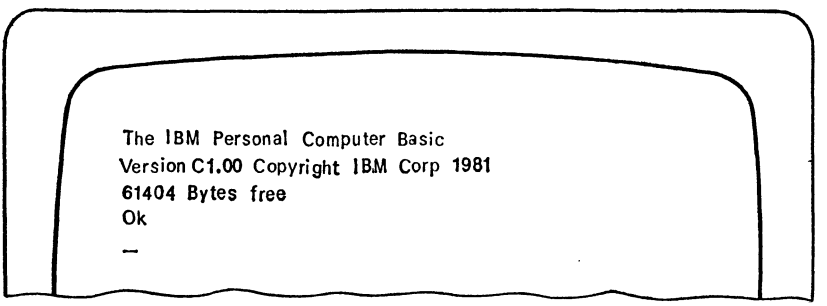
Рис. 3.3. Сообщения, выдаваемые сразу после успешной загрузки ДОС ПВМ.

путем нажатия комбинации клавиш **Ctrl|Alt|Del**. В обоих отмеченных случаях на дисководе А должна храниться программа ДОС ПВМ. На дискете с меткой DOS («Дисковая операционная система»), который поставляется в комплекте с руководством фирмы IBM, всегда записан один экземпляр ДОС; аналогичные копии ДОС могут храниться и на других дискетах. Если роль дисковода отведена разделу винчестерского диска, то экземпляр ДОС должен содержаться и на этом носителе. Диск с программой ДОС ПВМ называется *системным диском*.

При сбросе системы в исходное состояние самозагрузка ДОС ПВМ длится 10 с, а в случае включения системного блока 2 мин. Экран дисплея при этом очищается, и на нем появляется сообщение с просьбой ввести данные (рис. 3.3.А). В ответ на это сообщение набирают месяц, день и год, разделяя их дефисом или косой чертой, например 8-31-83 или 12/1/83. После ввода этих данных на экране высвечивается сообщение с просьбой ввести время (рис.



А *Диск с дефектом или не того типа*



В *Диск не вставлен или не закрыта крышка дисковода*

Рис. 3.4. Сообщения, выдаваемые после безуспешной попытки загрузить ДОС ПВМ.

3.3.В). Используя 24-часовую временну́ю шкалу, вводят часы, минуты и секунды, разделяя их двоеточиями, например 15:30:10 или 3 : 30 : 10. Секунды могут отображаться целым или дробным числом, но секунды, как, впрочем, и минуты, можно опустить. В этом случае их значение принимается равным нулю. Например, можно ввести 16:00:00 как 16:00 или просто как 16.

При появлении сообщения, отличного от запроса ввода данных, причину его следует искать в дискете, вставленном в дисковод А. Как только на экране высветится сообщение, начинающееся словами "Non-system disk" («Не системный диск») (рис. 3.4А), заменяют дискет в дисководе А на системный и нажимают клавишу пробела для повторной загрузки ДОС ПВМ. Появление в верхней строке экрана текста The IBM Personal Computer Basic (Бэйсик ПВМ фирмы IBM; рис. 3.4.В) означает, что крышка дисковода А не закрыта, либо дискет установлен не той стороной вверх, либо он вообще не предназначен для работы с ДОС ПВМ.

Возможны и иные сообщения об ошибках, а может оказаться и так, что дискет в дисководе А автоматически запустит какую-либо прикладную программу. В любом случае надо попытаться установить другой дискет.

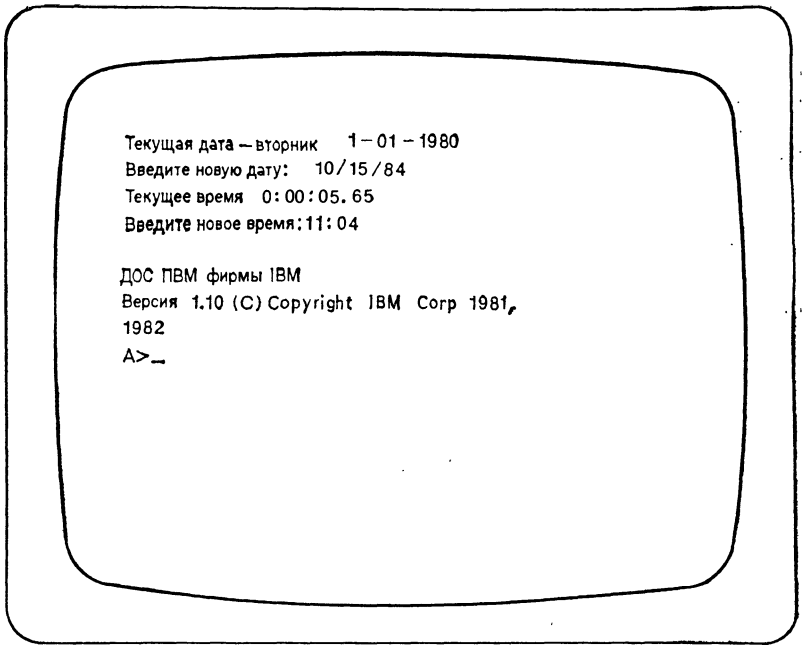


Рис. 3.5. Идентификационное сообщение, выдаваемое ДОС ПВМ.

После успешного ввода с клавиатуры времени суток ДОС ПВМ выводит на дисплей идентификационное сообщение и информацию об авторском праве (рис. 3.5). Символы A>, стоящие перед курсором, называются *командой подсказки*, поскольку они напоминают о том, что далее надо ввести какую-либо команду ДОС ПВМ.

Файлы ДОС ПВМ

Даже самый маленький дискет может иметь большую емкость, которая, однако, крайне редко используется целиком для какой-то одной цели. Обычно на одном диске сосуществует несколько маленьких независимых блоков информации, и чем больше его емкость, тем выше вероятность размещения на нем большого числа таких независимых блоков; винчестерский диск, например, способен хранить их в количестве нескольких сотен. Для организации хранения всех отдельных блоков информации в ДОС ПВМ используется нечто вроде картотеки: каждый диск рассматривается как один шкаф с делами, хранящимися в выдвижных ящиках, а каждый блок информации — как выдвижной ящик с одной папкой. Отдельные блоки информации на диске называются *файлами*.

Файлы, содержащиеся в памяти ЭВМ, классифицируются по виду хранимой информации, которая может представлять собой программы или данные. Программные файлы содержат команды, сообщающие машине, что и как сделать. Информационные файлы — это упорядоченные совокупности фактографических и числовых данных.

Имена файлов

Каждый дисковой файл имеет уникальное имя. Файлу может присваиваться любое еще не использованное имя при условии, что оно соответствует некоторому набору правил (рис. 3.6). Имя долж-

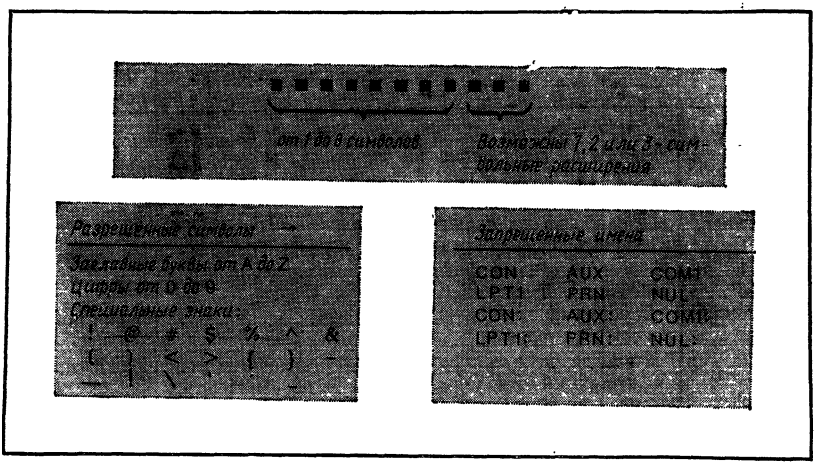


Рис. 3.6. Правила образования имен файлов ДОС ПВМ.

но содержать не менее одного символа; оно может состоять даже из восьми символов и снабжаться суффиксом. Необязательный суффикс, называемый *расширением имени файла*, состоит из точки, за которой следует не более трех символов.

Термин *имя файла* в точном его смысле относится только к корню, содержащему от одного до восьми символов, который предшествует расширению имени; однако обычно именем файла считается корень вместе с факультативным расширением, и именно такое определение будет использоваться в данной книге.

Для образования расширения имени файла разрешается применять отнюдь не любые символы и знаки (рис. 3.6). ДОС ПВМ одинаково манипулирует строчными и прописными буквами, поэтому имя файла можно набирать на клавиатуре в любом сочетании строчных и заглавных букв.

Если указанное имя файла состоит более чем из восьми символов и не содержит расширения, ДОС ПВМ автоматически ставит точку после восьмого символа, трактует следующие три символа как расширение и игнорирует остальные. Ошибка происходит в том случае, когда имя файла содержит более восьми символов перед явно указанным расширением; если расширение состоит более чем из трех символов, ДОС ПВМ игнорирует лишние.

◆◆◆ Общепринято, что расширение имени файла должно обозначать его тип. Чтобы не нарушать установленных соглашений и не вносить возмущений в работу других пользователей ДОС ПВМ, применяют стандартные расширения имен, перечисленные в табл. 3.1.

Таблица 3.1. **Расширения имен и типы файлов**

Расширение имени	Соответствующий тип файла
.ASM	Исходная программа на языке ассемблера
.BAK	Резервный файл или копия некоторого файла, сделанная в случае повреждения оригинала
.BAS	Программа на языке Бэйсик
.BAT	Командный файл для пакетной обработки
.BIN	Промежуточный файл для компилятора
.COB	Исходная программа на языке Кобол
.COD	Версия файла .OBJ на языке ассемблера (сгенерированная компилятором)
.COM	Команда или программа, пригодные для непосредственного исполнения под управлением ДОС ПВМ
.DAT	Файл данных
.DOC	Файл документов (для текстовой обработки)
.EXE	Перемещаемая программа, выполняемая под управлением ДОС ПВМ
.FOR	Исходная программа на языке Фортран
.HEX	Символьное шестнадцатеричное представление (в коде ASCII) двоичных данных
.LIB	Библиотека программ
.MAC	Макрокоманда для программы на языке ассемблера
.MAP	Листинг программы редактирования связей
.OBJ	Скомпилированная объектная программа на машинном языке
.OVL	Оверлейный файл прикладной программы
.OVR	Оверлейный файл программы компилятора
.PAS	Исходная программа на языке Паскаль
.PIC	Данные выводимого на экран изображения
.PRN	Листинг ассемблера
.SYM	Таблица символов для компилятора
.TER	Описание терминала (обслуживающая программа фирмы IBM для асинхронной передачи)
.TMP	Временный файл
.TXT	Текстовой файл
.$$$	Временный или неправильно хранимый, но пригодный для использования файл

Родовые имена файлов

В большинстве случаев желательно, чтобы присвоенные файлам имена были индивидуальными и единственными и точно указывали, какой именно файл имеется в виду. Но иногда удобнее обратиться сразу ко всей группе файлов, а не работать с ними по одному. Для указания родовых имен файлов, которые называются также *глобальными именами* или *неоднозначными именами* файлов, могут использоваться специально предназначенные для этого знаки ? и *.

Вопросительный знак является единственным неоднозначно интерпретируемым символом в имени файла. Вообще в операционной системе ДОС ПВМ смысл любого символа определяется индивидуально, но всякий одиночный символ считается соответствующим вопросительному знаку. Например, имена PHASE1.BAS, PHASE2.BAS, PHASE3.BAS и PHASE4.BAS все соответствуют родовому имени файла PHASE?.BAS, но ему не будут соответствовать имена PHASE10.BAS и PHASE153.BAS.

Звездочка обозначает любое количество неоднозначно интерпретируемых символов. Например, PGM*.B* будет обозначать любое имя файла, которое начинается с PGM, при условии, что его расширение начинается с B. Однако звездочка имеет смысл только при использовании ее в качестве последнего символа имени файла или расширения. Родовое имя файла *CALC.BAS — это то же самое, что *.BAS, которое соответствует каждому файлу, имеющему расширение имени BAS. Подобным же образом комбинации символов *.*K и *.* соответствуют любому имени файла.

Обозначения дисководов

Все файлы на одном диске должны иметь разные имена, но на разных дисках могут использоваться файлы с одним и тем же именем. Следовательно, в ПВМ с несколькими дисководами можно иметь два или более диска с одинаковыми именами файлов. Чтобы устранить неопределенность, перед именем файла ставят двухсимвольный префикс, обозначающий дисковод. Первый символ — это метка дисковода (A, B, C и т. д.), а второй символ — двоеточие. Таким образом, имя файла B:ADDRESS.DAT указывает на то, что этот файл находится на дисководе B.

В системе с одним дисководом существует только один дисковод A, и поэтому нет необходимости употреблять перед именами файлов префиксы: ДОС ПВМ будет автоматически обращаться к единственному доступному дисководу. Если же все-таки желательно указать дисковод, используют только обозначение A:.

Указание на какой-либо дисковод можно опустить и в системе с несколькими накопителями, так как ДОС ПВМ регистрирует дисковод, используемый по умолчанию. Этот дисковод называется *заре-*

гистрированным или *подразумеваемым*. Первоначально ДОС ПВМ регистрирует для использования по умолчанию дисковод А, что и указывается в наводящем сообщении A>.

Каталог имен файлов

Определенная часть каждого диска отводится для запоминания его «оглавления»: здесь содержится поименный перечень всех файлов, записанных на диске, и указывается их расположение. Ограничение по количеству файлов составляет для одностороннего дискета 64, а для двустороннего дискета 112. Винчестерские диски имеют переменные ограничения на предельное число файлов.

Команды ДОС ПВМ

ДОС ПВМ располагает многочисленными командами, с помощью которых можно поддерживать диски в соответствующем состоянии. Существуют команды, действие которых распространяется на диск в целом, на отдельные файлы, на каталог диска, на системное время и дату и т. д.

Формирование команд

Для указания ДОС ПВМ команд набирают определенные слова и символы на клавиатуре: набираемое сообщение указывает ДОС ПВМ, что делать и зачем. По мере набора команды ДОС ПВМ на клавиатуре текст отображается на экране дисплея. Чтобы ввести высвеченную на экране команду в машину, надо нажать клавишу ←┘; после этого ДОС ПВМ ее выполнит. В командных словах можно использовать любой набор строчных и прописных букв.

Резидентные и транзитные команды

Программа ДОС ПВМ содержит пошаговые инструкции, необходимые для выполнения многочисленных команд, называемых *резидентными* или *внутренними* командами. В случае ввода команды, которую ДОС ПВМ не может найти в своем наборе резидентных команд, операционная система разыскивает на диске файл с таким же именем, как имя введенной команды, и с расширением BAT, COM или EXE. Если нужный файл найден, операционная система пересылает инструкции из этого файла в динамическую память и выполняет их. Такие команды называются *транзитными* или *внешними* командами, так как инструкции для их выполнения не находятся постоянно в самой программе ДОС ПВМ. На системном дискете, поставляемом фирмой IBM в комплекте машины, пре-

дусмотрено несколько таких команд. Стандартные резидентные и транзитные команды перечислены в табл. 3.2.

Таблица 3.2. **Стандартные команды ДОС ПВМ**

Команда	Тип		Команда	Тип	
	резидентная	транзитная		резидентная	транзитная
CHKDSK		•	FORMAT		•
COMP		•	MODE		•
COPY	•		PAUSE	•	
DATE [1]	•		REM	•	
DIR	•		RENAME	•	
DISKCOMP		•	SYS		•
DISKCOPY		•	TIME [1]	•	
ERASE	•		TYPE	•	
EXE2BIN [2]		•			

[1] В ДОС ПВМ версии 1.0 команды DATE и TIME являются транзитными.
[2] Программисты, работающие с Бэйсиком, редко пользуются командой EXE2BIN, и поэтому в данной книге она не рассматривается.

Можно придумать и новую транзитную команду, создав файл, содержащий инструкции для ее выполнения. Существует также способ конструирования новых команд, при котором они комбинируются из уже имеющихся (гл. 5).

Как правило, ДОС ПВМ ищет транзитные команды на диске, объявленном по умолчанию. Чтобы явно указать, на каком именно дисководе ДОС ПВМ должна вести поиск, перед вводимой командой ставят префикс, являющийся обозначением дисковода. Если ДОС ПВМ не может найти инструкции для транзитной команды, она выводит на дисплей сообщение "Bad command or file name" («Неправильная команда или неверное имя файла»). В ответ на это еще раз набирают ту же самую команду, подтверждая, что файл, содержащий для нее инструкции, находится на указываемом дисководе.

◆◆◆ Следует проявлять особую осторожность в части смешивания команд ДОС версий 1.0 и 1.1. При наличии более чем одной версии ДОС ПВМ возникает рискованная ситуация: команды с одинаковыми именами могут казаться в разных версиях идентичными, а на самом деле внутренние инструкции для выполнения этих внешне идентичных команд могут существенно различаться. Можно было бы, например, загрузить версию 1.1 ДОС ПВМ с одного дискета, а затем заменить его дискетом с записью файлов транзитных команд версии 1.0. Но этого ни в коем случае делать нельзя, так как файлы могут разрушиться в результате выполнения несоответствующих им транзитных команд!! На каждом диске, содержащем файлы тран-

зитных команд, пишут имена файлов и номер версии, с которой они работают, тщательно избегая рассогласования файлов команд с загруженной версией операционной системы.

Исправление ошибок ввода с клавиатуры

Прежде чем нажать клавишу ↵ для выполнения команды, смотрят на экран дисплея и проверяют правильность того, что на нем высвечено. При обнаружении ошибки используют клавишу ←; это позволяет вернуться к началу того участка, в котором находится ошибка. Затем повторно набирают на клавиатуре команду, начиная от точки возврата. Если же команда испорчена настолько, что исправления себя не оправдывают, используют клавишу **Esc** для стирания всей строки. При нажатии клавиши **Esc** ДОС ПВМ выводит на экран символ \, пропускает напечатанную строку и перемещает курсор в начало следующей строки; наводящее сообщение при этом на экран не выводится.

Работа с печатающим устройством

Печатающее устройство может быть использовано для получения документальной копии команд ДОС ПВМ, вводимых с клавиатуры. Сделать это можно двумя способами. Нажимают одновременно клавиши ⌂ и **PrtSc**; ДОС ПВМ печатает один экземпляр текста, высвеченного на экране дисплея. Если одновременно нажать клавиши **Ctrl** и **PrtSc**, то ДОС ПВМ будет повторять каждую набираемую строку на печатающем устройстве, создавая синхронный протокол диалога. Для прекращения этого процесса нужно повторно нажать те же клавиши.

Обычно нажатием клавиши **PrtSc** активизируется только печатающее устройство параллельного действия, но можно сделать так, чтобы вместо этого устройства включалось печатающее устройство последовательного типа. Необходимая трансформация действия клавиши **PrtSc** осуществляется с помощью команды MODE.

Переназначение дисковода, объявляемого по умолчанию

Первоначально ДОС ПВМ обращается к дисководу A всякий раз, когда обозначение требуемого дисковода не указано явно. Чтобы использовать по умолчанию какой-либо другой дисковод, необходимо напечатать обозначающую его букву, затем двоеточие и нажать клавишу ↵. На экране дисплея появится новое наводящее сообщение ДОС ПВМ, говорящее о том, какой дисковод зарегистрирован теперь для обращения по умолчанию.

Пример. Переназначение подразумеваемого по умолчанию дисковода B.

```
A>b:
B>
```

Отмена команды

Для многих команд ДОС ПВМ требуется указывать явно обозначения дисководов или имена файлов. Если случайно был указан не тот дисковод или файл и ошибочную команду надо отменить, нажимают вместе клавиши **Ctrl** и **Scroll Lock**. Ответное сообщение ДОС ПВМ (например, A>) появится снова.

Просмотр каталога

Чтобы вывести на дисплей перечень файлов, записанных на диске, обозначаемом по умолчанию, используют команду DIR:

```
A>dir
```

В данном случае каталог диска, находящегося на накопителе A, выводится на экран после нажатия клавиши ↵ для выполнения указанной команды.

Каталог может содержать, однако, так много имен файлов, что его невозможно будет вывести на экран целиком. В подобной ситуации для постраничного вывода каталога на дисплей команда DIR снабжается суффиксом " /p":

```
A>dir /p
```

Каталог можно сжать так, чтобы на всей площади экрана появились в несколько столбцов только имена файлов и их расширения; для этого в команду DIR добавляют суффикс " /W":

```
A>dir /w
```

Анализ каталога

Каждая строка каталога состоит из пяти разделов: имени файла, расширения имени, размера файла, даты и времени последнего изменения файла (рис. 3.7). Отметим, кстати, что в каталоге точка между именем файла и расширением имени может быть опущена. Размер файла выражается числом байт, которое этот файл занимает на диске.

Каталог произвольного дисковода

Предположим, что требуется получить в виде списка данные каталога какого-либо определенного дисковода. Для этого достаточно поставить в конце команды DIR обозначение дисковода:

```
A>dir b:
```

С помощью такой команды выдается каталог диска, стоящего на дисководе B.

Заметим, что перед обозначением дисковода стоит пробел: если бы этого пробела не было, в задачу ДОС ПВМ входил бы поиск на

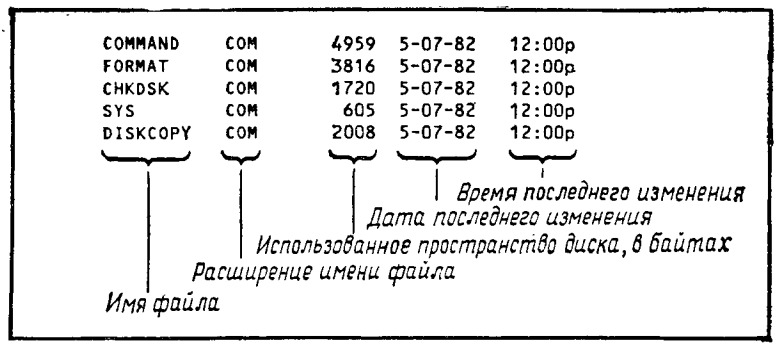


Рис. 3.7. Разделы каталога.

дисководе, подразумеваемом по умолчанию, транзитной команды DIRB:

Вместо использования команды с пробелом можно переназначить дисковод, принимаемый по умолчанию, с тем, чтобы он стал дисководом, содержащим диск с нужным каталогом:

```
A>b:
B>dir
```

Изменение принимаемого по умолчанию дисковода вызвано тем, что DIR — резидентная команда, и ДОС ПВМ известно, как эта команда должна исполняться независимо от имени подразумеваемого дисковода.

Избирательная выдача списков каталога

Команда DIR обеспечивает также и поиск нужного файла: для этого достаточно присоединить к команде его имя:

```
A>dir b:budget.bas
```

Представленная в такой форме команда DIR инициирует поиск в соответствующем каталоге указанного имени файла и выдает на экран соответствующий список при условии, что это имя существует. Если этот дисковод объявлен по умолчанию, указание конкретного дисковода перед именем файла необязательно.

Имя файла может быть собственным или родовым. В последнем случае по команде DIR выводятся на экран все имена файлов, соответствующие данному родовому имени.

Пример. В случае команды

```
A>dir *.com
```

выдаются все имена файлов с расширением COM, находящихся на дисководе A.

Подготовка пустых дисков

Новый диск, не содержащий никакой информации, начинает работать только тогда, когда его поверхность размечена должным образом, устранены выявленные дефекты и отведено конкретное место для каталога. Такой процесс инициализации диска называется *форматированием*; применительно к дискетным накопителям фирмы IBM он реализуется командой FORMAT.

Пример. В случае команды

```
A>format b:
```

размечается дискет, находящийся на дисководе B.

При нажатии клавиши ↵ появляется сообщение

```
Insert new diskette for drive B:
and strike any key when ready_
```

(Вставьте новый дискет в дисковод B:
и по готовности нажмите любую клавишу)

Дискет, подлежащий разметке, помещают в дисковод B, предварительно убедившись в том, что именно вложенный дискет подлежит форматированию. Здесь нужна особая осторожность, поскольку в процессе разметки дискета стирается вся записанная на нем информация.

◆◆◆ Никогда не подвергайте разметке единственный экземпляр какого-либо особо ценного для вас дискета!

Установив дискет, нажимают клавишу пробела или клавишу ↵. Этим действием инициируется разметка, и на экране появляется несколько пустых строк, за которыми следует сообщение «Форматирование...». В тот же момент на дисководе загорается лампочка, и он начинает работать, издавая ритмичный звук.

Форматирование одностороннего дискета занимает около 20 с; по окончании процесса появляется сообщение, подобное следующему:

```
Formatting . . . Format complete
160256 bytes total disk space
160256 bytes available on disk
Format another (Y/N)?_
(Форматирование... Форматирование завершено
Суммарная емкость диска 160256 байт
Свободно 160256 байт
Будет ли разметка очередного диска (Да/Нет)?)
```

После ответа на вопрос, поставленный в сообщении, нажимают клавишу Y (Да) или N (Нет).

◆◆◆ Ни в коем случае нельзя нажимать затем клавишу ←┘. Если это все же произойдет (или после Y будет нажата любая другая клавиша), ДОС ПВМ не станет ждать, пока будет вставлен новый дискет в указанный дисковод; она продолжит работу и переразметит заново дискет, который только что был форматирован. Никакого вреда это не принесет, но время будет потрачено впустую.

Команда FORMAT отмечает все поврежденные дорожки, которые она обнаруживает, предотвращая их выделение в будущем под какой-либо файл.

Односторонние и двусторонние дискеты

Команда FORMAT предусматривает автоматическое определение емкости дискета в указанном дисководе. В односторонних дискетах размечается только одна сторона в предположении, что эти дискеты будут использоваться в накопителях емкостью 160К или 320К байт. Если форматирование осуществляется с использованием дискового накопителя емкостью 160К, то размечается только одна сторона дискета, даже если сам дискет двусторонний.

В накопителе емкостью 320К форматируются обычно обе стороны двустороннего дискета. Добавление «/1» в конце команды FORMAT приводит к одностороннему форматированию независимо от типа дискета:

```
A>format b:/1
```

Именно проведенная разметка дискета, а не его потенциальные возможности определяет объем информации, который он может хранить, и с каким дисководом его можно использовать. Любой дискет, размеченный с одной стороны, хранит максимум 160К байт независимо от конкретного типа дисковода. Если же дискет двусто-

ронний, его переразметка для работы в накопителе емкостью 320К обеспечивает полное использование имеющихся возможностей, однако уже записанная на дискете информация будет, разумеется, стерта. Наоборот, дискет, размеченный с двух сторон, способен работать только в накопителе емкостью 320К. Любая попытка использовать его в дисководе емкостью 160К вызовет появление на экране сообщения

"Disk error reading drive A:."
("Ошибка чтения при работе с дисководом A:.")

Создание системных дисков

Напомним, что программа самозагрузки может загружать ДОС ПВМ только с системных дисков. Для формирования системного диска необходимо добавить суффикс «/S» к команде FORMAT:

```
A>format /s.
```

Этот необязательный суффикс указывает на необходимость хранения ДОС ПВМ на форматированном дискете, что требует создания в общей сложности трех файлов, обозначаемых IBMBIO.COM, IBMDOS.COM и COMMAND.COM. Два первых файла записываются таким образом, что их имена не попадают в каталог дискета, поэтому они называются *скрытыми файлами.*

Форматирование произвольного дисковода

Команда FORMAT относится к числу транзитных, поэтому в ней может требоваться префикс в виде обозначения дисковода.

Пример

```
B>a:format
```

Здесь по умолчанию подразумевается дисковод B, но, зная, что на этом дисководе нет файла FORMAT.COM, пользователь для файла транзитных команд предназначил дисковод A. Однако в приведенном примере диск B все равно будет форматироваться. Объясняется это тем, что указанное пользователем обозначение дисковода стоит не после командного слова, и поэтому команда FORMAT работает с дисководом, принимаемым по умолчанию, а таковым в данном случае является дисковод B.

Форматирование накопителей на винчестерских дисках

С некоторыми винчестерскими накопителями работает модифицированная команда FORMAT, которая обнаруживает повреждения и фиксирует любые дефектные участки, предотвращая их использование для хранения информации. Для большинства винчестерских дисков имеется отдельная команда, с помощью которой вся поверх-

ность диска разделяется на отдельные «накопители», и создаются пустые каталоги каждого из них. Однако ни одна из программ инициализации винчестерских дисков не похожа на другую, и поэтому необходимо тщательно ознакомиться с руководством по эксплуатации конкретного винчестерского накопителя, имеющегося в комплекте ПВМ.

Ошибки, препятствующие форматированию

Если при работе ДОС ПВМ возникают затруднения с разметкой дисков, на экране дисплея появляется сообщение, содержащее указание на возможный характер ошибки; например, дискет может быть повернут не той стороной, защищен от записи, поврежден или может быть вообще не того типа.

Дублирование дискетов

Одной из наиболее важных команд ДОС ПВМ является команда DISKCOPY, поскольку с ее помощью получаются дубликаты дискетов, которые могут быть использованы при отказе основного дискета. Эти дубли называются *резервными копиями*. Команда DISKCOPY работает только с накопителями на дискетах; для винчестерских накопителей имеется отдельная команда, с помощью которой можно выборочно получать копии файлов, изменявшихся уже после того, как было произведено последнее дублирование. Связанные с этим процессом конкретные операции столь разнообразны, что все их невозможно рассмотреть в пределах этой книги; в любом случае следует выполнять указания Руководства по эксплуатации конкретного дискового накопителя имеющейся у пользователя ПВМ.

Дублирование на двух дисководах

При наличии в комплекте ПВМ двух дисководов (каждый емкостью 160К или 320К) можно скопировать содержимое одного из них, называемого *исходным*, на другой, называемый *целевым*.

Пример. Содержимое накопителя A копируется на накопителе B.

```
A>diskcopy a: b:
```

Первым указывается исходный дисковод, вторым — целевой. В данном случае роль первого выполняет дисковод A, а второго — дисковод B, но можно использовать и любые другие разрешенные имена дисководов. При нажатии клавиши ↵ на экране дисплея появляется следующее сообщение:

```
Insert source diskette in drive A
Insert target diskette in drive B
Strike any key when ready
```

(Вставьте исходный дискет в дисковод A
Вставьте целевой дискет в дисковод B
По готовности нажмите любую клавишу)

◆◆◆ Дискеты необходимо устанавливать именно так, как указано в сообщении. Ни в коем случае нельзя менять местами исходный и целевой дискеты! В качестве дополнительной меры предосторожности следует заклеить на исходном дискете вырез, разрешающий запись (рис. 3.1). Если вы все-таки случайно перепутали дискеты, копирования не произойдет.

Проверив исходный дисковод и дискет, ДОС ПВМ определяет, какую копию нужно сделать: одностороннюю или двустороннюю, и выводит соответствующее сообщение на экран. Для успешного копирования двустороннего дискета на двух дисководах оба накопителя — и исходный, и целевой — должны иметь емкость 320К. Если, однако, в исходном дисководе используется односторонний дискет, то возможны любые комбинации накопителей разной емкости — 160К и 320К.

ДОС ПВМ копирует на целевой дискет всю информацию исходного, включая неиспользуемые его участки. В процессе копирования все, что было на целевом дискете, стирается. Копирование одностороннего дискета длится около 35 с. По окончании копирования на экране дисплея появляется сообщение "Copy complete" («Копирование завершено»).

Затем на экране высвечивается вопрос, надо ли копировать следующий дискет. Для ответа следует нажать одну из клавиш Y или N.

◆◆◆ Нельзя нажимать клавишу ←┘. Если все же это произойдет (или после Y будет нажата любая другая клавиша), ДОС ПВМ не станет ждать, пока вы вставите новый дискет в целевой дисковод. Вместо этого она продолжит работу и скопирует заново только что полученную информацию. Это не нанесет никакого ущерба, но время будет потрачено впустую.

«Дублирование» на одном дисководе

С помощью команды DISKCOPY можно получать резервные копии дискетов, используя всего один дисковод.

Пример

```
A>diskcopy
```

В данном случае производится копирование с любого дисковода на дисковод A.

Затем нажимается клавиша ←┘, и на экране высвечивается сообщение

Insert source diskette in drive A:
Strike any key when ready

—

(Вставьте исходный дискет в дисковод A:
По готовности нажмите любую клавишу)

Для безопасности перед вставлением исходного дискета в дисковод необходимо защитить его от случайной записи с помощью специальной наклейки. После нажатия любой клавиши ДОС ПВМ считывает с исходного дискета столько информации, сколько позволяет место в динамической памяти, и появляется сообщение

Insert target diskette in drive A:
Strike any key when ready

—

(Вставьте целевой дискет в дисковод A:
По готовности нажмите любую клавишу)

В ответ на это сообщение исходный дискет должен быть заменен целевым (на целевом дискете не должно быть наклейки, защищающей от записи). После нажатия любой клавиши ДОС ПВМ скопирует на целевой дискет информацию, только что считанную с исходного дискета. Цикл перестановки дискетов повторяется столько раз, сколько требуется для полного копирования исходного дискета.

Дублирование дисков [1]

По желанию при копировании можно опустить одно из обозначений дисководов. В этом случае явно указанный дисковод будет исходным, а подразумеваемый — целевым.

Пример

```
A>diskcopy b:
```

Ошибки при дублировании дисков

Если при работе ДОС ПВМ возникает какое-либо затруднение, связанное с копированием дискетов, на экран дисплея выдается сообщение, характеризующее возможную ошибку. Например, исходный дискет может быть повернут не той стороной, поврежден или может быть вообще не того типа. Может оказаться, что целевой дискет установлен неправильно или защищен от записи наклейкой, или не размечен, или поврежден либо не соответствует данному типу накопителя или вообще несовместим с исходным дисководом.

---

[1] Один из возможных способов дублирования дисков описывается в разделе гл. 5, посвященном копированию файлов.

При дублировании возможны также и не столь губительные ошибки, а само отсутствие сообщения об ошибке еще не гарантирует пригодности целевого дискета к работе. Получить такую гарантию можно только после формального сравнения исходного и целевого дисков или проверки состояния целевого дискета по окончании дублирования.

Сравнение дисков

При сравнении содержимого двух дискетов с помощью команды DISKCOMP используются один или два дисковода. Возможна любая комбинация одно- и двусторонних дискетов с учетом двух ограничений, о которых упоминалось выше.

Пример. Сравнение дискета в дисководе A с дискетом накопителя B.

```
A>diskcomp a: b:
```

Команда DISKCOMP выглядит так же, как команда DISKCOPY. В ней указываются два дисковода, подлежащих сравнению; это могут быть разные дисководы или один и тот же. В последнем случае ДОС ПВМ сообщает, когда необходимо переставить дискеты. Если опустить в явном виде обозначение дисковода, будет использован дисковод, принимаемый по умолчанию.

Когда команда DISKCOMP заканчивает сравнение, на экран выводится вопрос: «Сравнивать ли другие дискеты? (Да/Нет)». Утвердительный ответ дается при условии, что вы хотите произвести очередное сравнение с использованием тех же самых назначений дисководов.

Сравнение односторонних и двусторонних дискетов

ДОС ПВМ проверяет дисководы и дискеты, определяет, какое требуется сравнение — одной или двух сторон,— и выдает принятое решение на экран. Трудность возникает лишь тогда, когда первый дисковод двусторонний, а второй односторонний. В этом случае появляется сообщение "Incompatible diskette or drive types" («Несовместимые типы дискетов или дисководов»). Чтобы провести сравнение одной стороны дискетов, надо добавить в команду DISKCOMP символы «/1»:

```
A>diskcomp a: b: /1
```

Сведения о состоянии диска

Для анализа состояния дискета и его каталога используется команда CHKDSK, что обеспечивает их внутреннюю целостность и согласованность:

```
A>chkdsk b:
```

В приведенном примере контролируется дисковод B. Если опустить обозначение дисковода, проверяться будет дисковод, принимаемый по умолчанию.

Анализ начинается сразу после ввода команды; при этом нет паузы для замены дискетов. По окончании проверки выдается сообщение, подобное следующему:

```
160256 bytes total disk space
  8704 bytes in 2 hidden files
144384 bytes in 26 user files
  7168 bytes available on disk
 65536 bytes total memory
 53136 bytes free
```

(Суммарная емкость диска 160256 байт
2 скрытых файла 8704 байт
26 пользовательских файлов 144384 байт
Свободно на диске 7168 байт
Общий объем памяти 65536 байт
Свободно 53136 байт)

В сообщении указывается емкость диска (160K), количество скрытых файлов (если таковые есть) и объем памяти, который они занимают; число пользовательских дисковых файлов и их объем, а также свободный объем дисковой памяти. Если на диске есть дефектные дорожки или секторы, дополнительно сообщается о занимаемом ими пространстве. Наконец, в сообщении указывается общий объем динамической памяти ЭВМ и та ее часть, которая остается после размещения программы ДОС ПВМ.

В процессе анализа при выполнении команды CHKDSK проверяемый дисковод временно считается подразумеваемым по умолчанию, поэтому, если завершить анализ преждевременно (например, путем нажатия комбинации клавиш **Ctrl|Scroll Lock**), необходимого переназначения дисковода по умолчанию не произойдет.

Настройка границ рабочей области экрана

Большинство бытовых телевизоров, используемых в качестве дисплеев вычислительных машин, не способно обеспечивать посимвольное отображение информации в каждой строке. Они увеличивают принимаемое изображение таким образом, что крайние левые или крайние правые элементы либо и те и другие вместе оказываются вне поля видимости. Компенсировать эффект развертки за пределами экрана можно с помощью команды MODE, которая позволяет сдвигать изображение вправо или влево.

Пример. Сдвиг изображения вправо.

```
A>mode 40,r,t
```

В ответ на данную команду ДОС ПВМ помещает в верхней части экрана 40-символьный контрольный шаблон, сопровождаемый вопросом: "Виден ли левый крайний 0? (Да/Нет)"

```
01234567890123456789012345678901234567890123456789
Do you see the leftmost 0? (Y/N)
```

Если нуль у левой границы экрана виден, нажимается клавиша «**Y**», в противном случае — клавиша «**N**», после чего изображение сдвигается на один символ вправо. Клавиша «**N**» должна оставаться нажатой до тех пор, пока крайний слева 0 не появится в поле зрения.

Команда MODE имеет несколько модификаций, и вместо сдвига вправо, как в приведенном выше примере, с помощью этой команды можно осуществить сдвиг влево:

```
A>mode 40,l,t
```

В данном случае опять появится на экране прежний контрольный шаблон, но при ответном нажатии клавиши «**N**» изображение сместится влево.

Единственное число 40, которое использовалось до сих пор в примерах, устанавливает ширину экрана дисплея, равную 40 символам. Можно задать и 80-символьную строку, используя число «80» вместо «40», однако большинство телевизионных приемников не обладает достаточной разрешающей способностью для четкого отображения 80 символов в одной строке.

Команда MODE, в которой указано только число, характеризующее ширину экрана, будет лишь настраивать экран на указанную длину строки. Само изображение при этом сдвигаться не будет. Можно вообще не указывать в команде MODE число (40 или 80), тогда ширина рабочего поля экрана будет всегда оставаться неизменной. Если опустить и первое число, и последний символ T, чтобы подавить вывод на экран контрольного шаблона, то сразу после ввода команды MODE изображение будет сдвинуто на один символ при 40-символьной длине строки и на два символа в случае 80-символьной строки. Три рассмотренные модификации команды задания режима работы экрана имеют вид

```
A>mode 40
A>mode l,t
A>mode ,r
```

Команда MODE не оказывает никакого воздействия на монохроматический дисплей: она работает только с дисплеями, подключенными к адаптеру цветного монитора и графических устройств.

Выбор характеристик печатающего устройства

Существуют три модификации команды MODE, работающие не с экраном дисплея, а с печатающим устройством. Пользуясь этими разновидностями команды, можно изменять длину строки, задавать число строк на странице и менять режим информационного обмена ПВМ с печатающим устройством с параллельного на последовательный.

Изменение интервала печати

Модификация команды MODE, обеспечивающая возможность изменения интервала между символами и строками, рассчитана на работу с матричным печатающим устройством 80CPS фирмы IBM, печатающим устройством MX-80 фирмы Epson и с любыми другими печатающими устройствами, которые совместимы с указанными. Используемое печатающее устройство должно быть предварительно включено и подготовлено к печати до подачи следующей команды:

```
A>mode lpt1:132,8
```

После нажатия клавиши ↵ появляется сообщение, подтверждающее прием введенной команды. Если появится сообщение "Printer error" («Ошибка печати»), то скорее всего печатающее устройство просто не готово к работе.

Первый компонент рассматриваемой модификации команды MODE, имеющий мнемонику LPT1:, сообщает ДОС ПВМ, что команда относится к печатающему устройству. Второй компонент указывает число символов в строке, для которого допустимы только два значения: 80 или 132. В каждом случае длина строки остается постоянной, изменяется лишь размер символов. Третий компонент определяет число строк в 1 дюйме (здесь также возможны только два варианта: 6 или 8). Каждый раз, когда печатающее устройство включается, оно самонастраивается на печатание 80 символов в строке и 6 строк на один дюйм длины листа бумаги.

Применение печатающих устройств последовательного действия

ДОС ПВМ пересылает данные, выводимые на печать, в буфер печатающего устройства параллельного действия. Однако, формируя две команды MODE, можно адресовать эти данные к последовательному печатающему устройству. Тогда первая из команд согласовывает работу ПВМ с характеристиками печатающего устройства, а вторая реализует переадресацию выводимой информации. Это изменение действует до тех пор, пока оно не будет отменено в результате действия другой команды MODE или перезагрузки операционной системы.

Для организации последовательной связи в ДОС ПВМ используется распространенный стандарт RS232, обладающий гибкими возможностями, вследствие чего отправитель и получатель сообщений либо должны принять соглашение об условиях разрешения определенных конфликтных ситуаций, либо они вообще не смогут поддерживать связь. Команда MODE обеспечивает соответствие условий работы ДОС ПВМ требованиям со стороны внешнего устройства последовательного действия.

Для этого пользователю прежде всего необходимо определить условия, диктуемые последовательным печатающим устройством ПВМ. В табл. 3.3 приведены четыре характеристики режима после-

Таблица 3.3. **Характеристики стандарта последовательной передачи данных RS232, учитываемые командой MODE** [1]

Характеристика	Возможные значения	Тип, указываемый в команде	Назначение
Скорость передачи (в бодах)	110	11	Установление скорости передачи данных
	150	15	
	300	30 [2]	
	600	60	
	1200	12 [2]	
	2400	24 [3]	
	4800	48	
	9600	96	
Способ контроля	Без контроля	n [2]	Обнаружение ошибок
	По нечетности	o	
	По четности	e [3]	
Длина слова (в битах)	7	7 [3]	Длина каждого информационного байта
	8	8 [2]	
Число стоповых битов	1	1 [2]	Признак конца байта данных
	2	2	

[1]) Для печатающих устройств последовательного действия предусмотрена общая команда вида MODE COM1: *скорость передачи в бодах, способ контроля, длина слова, стоповые биты*, P. Для других устройств последовательного действия опускаются конечные символы «P».

[2]) Обычный вариант для последовательных печатающих устройств.

[3]) Эти значения задаются при первичной загрузке ДОС ПВМ.

довательной связи и возможные варианты, которыми располагает ДОС ПВМ по каждой из них. В руководстве по эксплуатации конкретного устройства печати следует найти такие его характеристики, как скорость передачи в бодах, способ контроля передачи, длина слова и число стоповых битов. При этом необходимо выяснить, нуж-

но ли для выбора конкретных режимов и характеристик устанавливать в соответствующее положение внутренние переключатели печатающего устройства.

Для перевода требуемых значений характеристик последовательной связи с печатающим устройством в обозначения, воспринимаемые командой MODE, удобно пользоваться табл. 3.3. Как следует из этой таблицы, команда

```
A>mode com1:30,n,8,1,p
```

настраивает ДОС ПВМ на пересылку символов со скоростью 300 бод, без контроля по четности, при длине слова 8 бит и одном стоповом бите.

После нажатия клавиши ↵ появляется сообщение, подтверждающее ввод.

Первый компонент этой модификации команды MODE, имеющий мнемонику COM1:, означает, что команда относится к режиму последовательной связи. Последний компонент, буква p, указывает ДОС ПВМ на то, что последовательная связь должна быть установлена именно с печатающим устройством, а не с каким-либо другим.

Команда MODE, реализующая переадресацию выводимой информации последовательному печатающему устройству, выглядит так:

```
A>mode lpt1:=com1
```

В результате ввода такой команды появляется сообщение "LPT1: redirected toCOM1:"

Для обратного переключения с режима последовательной передачи на параллельный используется команда

```
A>mode lpt1:
```

Копирование файлов

Для копирования отдельных файлов служит команда COPY в самых различных вариантах. Можно копировать файл, используя один или два дисковода любого типа — с дискетами на 160К или 320К, либо задавая в качестве накопителей разделы винчестерского диска. Имена исходного файла и дубля могут быть разными или, если используются два дисковода, одинаковыми. Возможно также использование команды COPY для пересылки выбранных файлов транзитных команд с дискета, содержащего операционную систему, на другой дискет. Дублируя только нужные файлы, пользователь экономит место на диске для других файлов.

Вариант одинаковых имен и разных дисководов

В простейшей форме записи команды COPY в ней указывается только имя исходного файла и это же имя присваивается копии файла. Исходный и целевой дисководы в этом случае должны быть разными.

Пример. Копирование файла с дисковода A на дисковод B.

```
A>copy a:mode.com b:
```

Имя исходного файла всегда стоит в команде COPY первым; в данном примере это имя A:MODE.COM. Символ A: обозначает исходный дисковод; обозначение B относится к целевому дисководу.

Операция копирования начинается сразу после нажатия клавиши ↵. При этом не появляется никакого сообщения и нет паузы, позволяющей заменить дискеты, так что все приготовления должны быть закончены до начала операции копирования. Команда COPY является в ДОС ПВМ резидентной, поэтому файл команд для нее не нужен. По окончании операции копирования на экран выдается сообщение, указывающее количество скопированных файлов.

Если имена исходного и целевого файлов совпадают, можно опустить обозначение либо исходного дисковода, либо целевого, но не обоих сразу. В качестве дисковода, обозначение которого не указано явно, в данном случае используется дисковод, принимаемый по умолчанию, однако при этом явно определяемый дисковод должен быть другим.

Пример. Два случая применения команды COPY, в каждом из которых производится копирование с дисковода A на дисковод B.

```
A>copy mode.com b:
        1 File(s) copied
(Скопировано файлов — 1)
A>b:
B>copy a:mode.com
        1 File(s) copied
(Скопировано файлов — 1)
```

В первом случае копирование осуществляется с дисковода, принимаемого по умолчанию, на дисковод, указанный в команде. Затем подразумеваемым становится дисковод B, после чего, во втором случае, копирование происходит с явно указанного дисковода на новый, принимаемый по умолчанию.

Если файл с тем или иным именем уже существует на целевом дисководе, он будет стерт и заменен файлом с исходного дисковода. При ошибочном копировании какого-нибудь файла на самого себя (когда одинаковы и имя файла, и обозначение дисковода) команда COPY сообщит о том, что скопировано 0 файлов, и никаких опасных последствий не будет.

Вариант разных имен и одного дисковода

Для того чтобы сдублировать отдельные файлы на одном дисководе, имена исходного и целевого файлов должны быть разными.

Пример

```
A>copy letter1.doc letter1.bak
```

Здесь резервная копия файла LETTER1.DOC получается на том же дисководе; при этом используется дисковод, принимаемый по умолчанию, однако может быть явно указан и любой другой.

Вариант разных имен и разных дисководов

Для создания дублирующего файла с другим именем и на дисководе, отличном от исходного, необходимо указать имена обоих файлов и обозначение хотя бы одного дисковода.

Пример

```
A>copy contract.doc b:contract.bak
```

В данном примере с файла CONTRACT.DOC, находящегося на подразумеваемом дисководе A, снимается копия CONTRACT.BAK, помещаемая на дисковод B. Обозначение исходного дисковода может указываться явно.

Применение команды COPY при работе с родовыми именами файлов

Команда COPY допускает использование неопределенных символов ? и * в именах файлов. Это позволяет ускорить работу за терминалом при условии, что имена исходного и целевого файлов совпадают, за исключением расширений.

Пример

```
A>copy amortize.bas *.bak
```

Используя родовые имена, можно скопировать все файлы исходного дисковода на целевой.

Пример. Копирование всех файлов дисковода A, принимаемого по умолчанию, на дисковод B может быть выполнено по команде

```
A>copy *.* b:
```

Копирование всех файлов с одного дисковода на другой с помощью команды COPY в основном аналогично получению копий с помощью команды DISKCOPY, хотя имеется и ряд существенных отличий. Прежде всего команда DISKCOPY не оставляет на целевом дисководе ни одного ранее записанного файла, а команда COPY добавляет новые файлы к уже записанным. Если же пространство, имеющееся на целевом диске, слишком мало для записи всех исход-

ных файлов, команда COPY скопирует лишь столько файлов, сколько можно.

Если диск перед началом работы команды COPY пуст, каталог диска эффективно реорганизуется в ходе ее выполнения. Благодаря такой реорганизации уменьшается время доступа к диску, особенно в тех случаях, когда на диске многократно создавались и уничтожались файлы. Поскольку каталог диска реорганизуется, весьма вероятно, что команда DISKCOMP объявит целевой дискет, на который информация записана с помощью команды COPY*.*, отличным от оригинала. Что же касается команды DISKCOPY, то она не реорганизует каталог и потому создает точный дубликат исходного диска.

Копирование с контролем

Если желательно, чтобы ДОС ПВМ осуществила считывание целевого файла, его сравнение с исходным и проверку их идентичности, то в конце команды COPY следует поставить суффикс "/V":

```
A> copy a:format.com b:/v
```

Ошибки при копировании файлов происходят редко и обусловлены обычно неисправностью целевого дискета. С суффиксом "/V" команда COPY работает медленнее, поскольку после записи целевого файла выполняется его проверка.

Стирание файлов

По команде ERASE имена одного или более файлов изымаются из каталога, а место, которое эти файлы занимали на диске, освобождается для других файлов. Команда работает с дисководами любого типа и выглядит следующим образом:

```
A>erase b:invntry.bak
```

Так же, как и в других командах, явное указание обозначения дисковода не обязательно, поскольку возможно использование принципа умолчания.

Команда ERASE может работать и с родовыми именами файлов, однако в этом случае необходимо проявлять осторожность, так как родовые имена используются главным образом тогда, когда группу файлов нужно уничтожить одной командой; поэтому даже незначительная ошибка в родовом имени файла может привести к стиранию совсем не той группы файлов.

Пример. Команда

```
A>erase b: *.*
```

обеспечивает стирание всех файлов на дисководе B, но предварительно к пользователю адресуется вопрос: "Are you sure (Y/N)?" («Вы уверены? (Да/Нет»)) в целях предотвращения случайного стирания всех файлов.

Переименование файлов

С помощью команды RENAME изменяется имя одного или более файлов на дисководе любого типа. До переименования новое имя файла не должно присутствовать в каталоге диска.

Пример

```
A>rename b:maillist.bas mail.bas
```

Первым указывается старое имя файла, а вторым — новое. Префикс перед первым именем, явно указывающий обозначение дисковода, не обязателен. Если его нет, значит, происходит изменение имени файла, находящегося на дисководе, принимаемом по умолчанию. Любое явное обозначение дисковода перед вторым именем файла командой RENAME игнорируется.

Рассматриваемая команда допускает использование родовых имен файлов.

Если старый файл имеет родовое имя, то все соответствующие ему файлы переименовываются согласно новому имени, которое тоже должно быть родовым.

Пример. Команда

```
A>rename 5*.* 4*.*
```

обеспечивает переименование всех файлов, имена которых начинаются цифрой 5, таким образом, что цифра 5 заменяется в них на 4.

Новое родовое имя файла, используемое в сочетании с конкретным старым именем, может существенно сократить объем информации, набираемой на клавиатуре.

Пример

```
B>rename address1.dat *.bak
```

В данном случае эффективно реализуется изменение расширения имени файла ADDRESS1 с DAT на BAK.

Глава 4

ЗАПУСК ПРОГРАММЫ

По мнению многих пользователей ПВМ, им никогда не придется самим писать программы для вычислительной машины, поскольку можно использовать уже существующие. По-видимому, в ряде случаев такое мнение вполне оправданно, особенно если учесть разнообразие имеющихся в настоящее время готовых прикладных программ и их высокое качество.

В данной главе описываются только самые общие команды, необходимые для запуска программ, входящих в состав серийных средств программного обеспечения. Эти команды, однако, ни в коей мере не заменяют специфических команд, связанных с каждой отдельной программой. Процедуры работы с различными прикладными программами настолько отличаются друг от друга, что подробно описать их все в одной этой главе практически невозможно.

Как правило, сразу после запуска прикладной программы на экран дисплея выводятся все специфические команды, которые достаточны для организации работы с данной программой. Подробная информация об этих командах содержится в руководстве для программистов; при необходимости можно проконсультироваться с теми, кто уже освоил работу с интересующей программой.

В основном прикладное программное обеспечение для ПВМ представлено в виде программ, записанных на дискетах либо напечатанных в книгах или журналах. Для того чтобы программа, находящаяся на любом из этих носителей, могла начать выполняться, ее необходимо предварительно поместить в динамическую память ПВМ. В случае когда программа записана на дискете, для этого достаточно всего лишь несколько раз нажать нужные клавиши, так как с помощью специальной аппаратуры информация может передаваться в память ПВМ непосредственно с дискетов. Если же программа представлена в виде текста, напечатанного в книге или журнале, то потребуется выполнить достаточно трудоемкую и утомительную работу — вводить программу в вычислительную машину вручную, так как в настоящее время пока еще нет средств непосредственного ввода печатного текста в ЭВМ.

Программное обеспечение на дискетах

Существуют различные способы запуска программы, записанной на дискете. Решение вопроса о том, какой именно способ следует выбрать для конкретной программы, зависит от многих факторов, и мы будем недалеки от истины, если скажем, что единственный способ выбрать подходящий метод состоит в том, чтобы поочередно применять все методы, пока не найдется работающий в данной ситуации.

Дублирование программного обеспечения

Чтобы избежать неприятностей (иногда достаточно серьезных), которые могут возникнуть в случае повреждения или неисправностей дискета, препятствующих успешному запуску программы, необходимо делать копии каждого вновь приобретенного дискета с про-

граммным обеспечением. Команды, предназначенные для копирования как всего дискета целиком, так и отдельных программ, были описаны в гл. 3.

Заметим, что некоторые пакеты прикладных программ записаны на дискетах таким образом, что дублирование их невозможно. В подобной ситуации следует предварительно выяснить, каким образом в случае необходимости можно получить копию такого дискета.

Роль дисковой операционной системы

Для перезаписи прикладной программы с дискета в динамическую память ПВМ необходима дисковая операционная система. Существует несколько таких систем, предназначенных для работы на ПВМ; в их число входят CP/M-86, UCSD Pascal (p-система), OASIS и наиболее широко используемая ДОС ПВМ. Приобретаемый дискет с программным обеспечением можно, как правило, использовать только при наличии определенной операционной системы (одной из указанных). Наиболее популярные пакеты прикладных программ имеют несколько версий, совместимых с различными операционными системами. При покупке программ необходимо удостовериться, что они предназначены именно для той дисковой операционной системы, которая имеется в вашем распоряжении. Мы ограничимся описанием процедуры запуска прикладных программ стандартного программного обеспечения в случае операционной системы ДОС ПВМ; аналогичные процедуры осуществляются и для других операционных систем.

Иногда приобретенный дискет с прикладным программным обеспечением не содержит копии дисковой операционной системы, что связано с лицензионными ограничениями, накладываемыми владельцами операционных систем. Но дисковая операционная система может копировать саму себя на любой выбранный дискет. Необходимые для этого команды применительно к ДОС ПВМ описываются в гл. 5. В целях предосторожности при дублировании операционной системы лучше использовать копию приобретенного дискета, а не оригинал.

Программное обеспечение со средствами автоматического запуска

Некоторые пакеты программного обеспечения могут запускаться автоматически. Все, что требуется сделать для запуска таких программ,— это поставить соответствующий дискет на дисковод A, закрыть дверцу дисковода и включить вычислительную машину. Если системный блок включен, то программы можно запустить нажатием клавиш **Ctrl|Alt|Del**. В обоих случаях процедура запус-

ка аналогична процедуре загрузки ДОС ПВМ, описанной в гл. 3, с той лишь разницей, что просто системный дискет заменен на дискет с автоматически запускаемыми прикладными программами.

Вскоре после указанных действий должен послышаться шум, вызванный работой дисковода А, и на экране дисплея начнут появляться различные символы: высветится по крайней мере одна команда ДОС ПВМ, а возможно, и больше. Эти команды поступают из специального файла с именем AUTOEXEC.BAT, содержащего все команды, необходимые для автоматического запуска прикладной программы [1]. В результате управление ПВМ будет передано прикладной программе и на экране появится ее первое сообщение. Описание специальных команд, предназначенных для работы с каждой отдельной программой, можно найти в руководстве по использованию прикладного программного обеспечения.

Ошибки, возникающие при автоматическом запуске программ

Вывод новых сообщений на экран дисплея может прекратиться еще до того, как управление ПВМ перейдет к прикладной программе. Это может быть связано с тем, что прикладная программа просто не является самозапускающейся. Если картина на экране дисплея такая же, как при загрузке ДОС ПВМ (см. рис 3.5), то необходимо начать запуск программы вручную.

Сообщение "Non-system disk or disk error" («Не системный диск или ошибка по вине диска») может означать, что на дискете с прикладными программами нет копии ДОС ПВМ. В этом случае можно либо скопировать ДОС ПВМ на этот дискет, либо запустить ее вручную. Указанное выше сообщение, так же как и сообщение "Disk error reading drive A" («Ошибка считывания с диска А»), может быть обусловлено неисправностями дискета. Однако, прежде чем делать подобный вывод, следует еще раз проверить, какой дискет установлен на дисководе А и правильно ли он установлен, а затем вновь нажать клавиши **Ctrl|Alt|Del**, т. е. повторить автоматический запуск. При появлении того же самого сообщения об ошибке, надо попробовать поставить другую копию дискета с прикладными программами. Если и после смены дискета ошибка остается неисправленной, то тогда действительно можно сделать предположение о сбое оборудования. На этот случай в технической документации фирмы IBM "Guide to Operations" («Руководство по операциям») содержится диагностический дискет и описываются команды, с помощью которых можно использовать этот дискет для проверки конкретной системы.

---

[1] Более подробно о файле AUTOEXEC.BAT см. гл. 5.

Запуск программ вручную

Прежде чем начать запуск прикладной программы, необходимо произвести загрузку дисковой операционной системы так, как это описывается в гл. 3. В результате загрузки на экране дисплея должно появиться идентификационное сообщение ДОС ПВМ, в конце которого после наводящей команды должен высветиться курсор (см. рис. 3.5). Вслед за этим необходимо определить, какое имя имеет данная прикладная программа на дискете. Имя может не совпадать с полным названием программы, а являться некоторой абревиатурой или акронимом. Чтобы определить имя программы, можно посмотреть документацию, прилагаемую к данному дискетному пакету программ, и найти рубрику «имя программы» или «имя файла». Найденное имя должно состоять не более чем из восьми символов и может оканчиваться точкой, за которой следуют три символа, например COM, EXE или BAS. Такое окончание указывает на тип файла, в котором хранится программа, а тип в свою очередь определяет, с помощью каких команд должен осуществляться запуск программы. Если имя программы и тип файла не удается определить каким-либо независимым способом, можно использовать команду ДОС ПВМ DIR (гл. 3) и с ее помощью просмотреть справочник имен файлов дискета с прикладным программным обеспечением.

Запуск прикладной программы типа COM или EXE

Для запуска программы, хранящейся в файле, имя которого оканчивается на COM или EXE, достаточно набрать на клавиатуре имя этого программного файла, а затем нажать клавишу ←┘. При этом нет необходимости набирать само окончание имени, поскольку указанные окончания (COM или EXE) приняты в ДОС ПВМ по умолчанию.

Пример. Запуск программы, записанной в файле WS.COM.

```
A>ws
```

Если имя файла введено правильно, то через несколько секунд после этого на экране дисплея должно появиться первое сообщение запускаемой программы. В случае появления сообщения об ошибке необходимо еще раз проверить, правильно ли было набрано имя. Если ошибки при наборе не допускались, а на экране появилось сообщение об ошибке "Bad command or file name" («Неправильная команда или неверное имя файла»), то это может означать, что либо запускаемой программы нет на данном носителе, либо она имеет другое имя, либо содержащий ее файл не является файлом типа COM или EXE.

Дискет с прикладными программами не обязательно устанавливать на дисковод A: чтобы при запуске в явном виде указать исполь-

зуемый дисковод, достаточно поставить перед именем программного файла идентификатор этого дисковода.

Пример

B>a:ws

В данном примере, несмотря на то что принятым по умолчанию является дисковод B, система ДОС ПВМ будет искать файл с именем WS.COM на дисководе A, поскольку последний был указан в явном виде.

Запуск прикладных программ типа BAS

Файлы типа BAS содержат программы, написанные на языке Бэйсик. Для выполнения этих программ необходим интерпретатор, транслирующий Бэйсик-команды в команды машинного языка ПВМ. Интерпретатор занимает часть динамической памяти ПВМ, где также находятся ДОС ПВМ и транслируемая программа.

В условиях ДОС ПВМ можно использовать две версии Бэйсика: дисковый и расширенный. Интерпретатор расширенного Бэйсика рассчитан на обработку более широкого набора команд, чем интерпретатор дисковой версии, но он занимает и больший объем динамической памяти, оставляя меньше места для прикладной программы. Если заранее неизвестно, какая версия интерпретатора требуется для запускаемой программы, то сначала следует попробовать интерпретатор расширенного Бэйсика. Этот интерпретатор не пригоден лишь в том случае, когда программа, написанная на дисковом Бэйсике, слишком велика, чтобы поместиться в памяти вместе с интерпретатором расширенной версии. Но подобная ситуация выявится сразу же, до запуска прикладной программы. Попытка использовать интерпретатор дискового Бэйсика может привести к нежелательным последствиям, если окажется, что в действительности для программы требовался интерпретатор расширенного Бэйсика. Такая программа будет работать в течение некоторого времени, пока наконец не встретится команда, входящая только в расширенный Бэйсик. После этого выполнение программы прекратится и разобраться в ее текущем состоянии будет практически невозможно.

Интерпретатор представляет собой некоторую программу, которая также должна быть перенесена с диска в память ПВМ. Интерпретатор дискового Бэйсика находится в программном файле с именем BASIC.COM, а расширенного — в файле с именем BASICA.COM.

Для запуска прикладной программы типа BAS сначала следует набрать на клавиатуре имя файла, содержащего нужный интерпретатор, пропустить один пробел и набрать имя файла, содержащего нужную прикладную программу, после чего нажать клавишу ←┘. При наборе имен файлов можно опустить все окончания: в ДОС ПВМ по умолчанию принято, что имя файла, содержащего интерпретатор,

оканчивается на COM, а имя файла с прикладной программой — на BAS.

Пример. Запуск программы, находящейся в файле SAMPLES.BAS, вместе с интерпретатором дискового Бэйсика.

```
A>basic samples
```

После нажатия клавиши ↵ под управлением ДОС ПВМ производится пересылка интерпретатора с диска в память ПВМ и очищается экран дисплея. Затем с диска в память ПВМ передается прикладная программа, и управление переходит к интерпретатору, который начинает обрабатывать команды прикладной программы.

В приведенном выше примере предполагалось, что и файл с интерпретатором, и файл с прикладной программой находятся на дисководе, принятом по умолчанию. Однако любой из этих файлов (или оба сразу) можно поместить и на другой дисковод, а при запуске указать этот дисковод в явном виде. Для этого достаточно просто поставить перед именем соответствующего файла идентификатор дисковода.

Пример

```
B>a:basica samples
```

В данном примере принятым по умолчанию является дисковод B, но символы, предшествующие имени файла с интерпретатором, говорят о том, что ДОС ПВМ должна искать интерпретатор расширенного Бэйсика на носителе A. Поскольку перед именем программного файла SAMPLES.BAS нет никакого идентификатора дисковода, этот файл должен находиться на дисководе, принятом по умолчанию.

В тех случаях, когда ДОС ПВМ не может найти Бэйсик-интерпретатор на указанном для него носителе, на экран дисплея выводится сообщение об ошибке "Bad command or file name" («Неправильная команда или неверное имя файла»). Если на соответствующем носителе не будет обнаружена прикладная программа, то после вывода 3 строк идентификационного сообщения интерпретатора на экране появится сообщение "File not found" («Файл не найден»), а под ним вновь повторится наводящее сообщение системы ДОС ПВМ.

Программы на винчестерских накопителях

Практически любую прикладную программу, записанную на дискете, можно скопировать на винчестерский накопитель и запускать ее оттуда точно так же, как и в случае обычного дискетного накопителя. Часто винчестерский накопитель имеет такой объем, который делает возможной запись нескольких пакетов прикладных программ. Зная имя содержащего ее файла, можно копировать про-

грамму с одного носителя на другой (гл. 3). Прикладная программа может занимать несколько файлов, поэтому нужно внимательно следить за тем, чтобы с дискета прикладных программ на винчестерский диск были скопированы все необходимые файлы.

Если накопитель А, входящий в состав используемой системы, является винчестерским, то одна из хранимых на нем прикладных программ может быть самозапускающейся. Выбрав из пакета прикладного программного обеспечения файл AUTOEXEC.BAT, который предполагается использовать как автоматически запускаемый, следует скопировать его на носитель А. Поскольку все файлы, записанные на одном и том же носителе, должны иметь уникальные имена, на носителе А может храниться только один файл с именем AUTOEXEC.BAT. Это в свою очередь означает, что если используется винчестерский диск, то самозапускающейся может быть только одна программа.

Использование прикладных программ, представленных в виде напечатанных текстов

Различные книги и журналы содержат тексты Бэйсик-программ, предназначенных для работы на ПВМ. В настоящее время существует лишь один способ ввода напечатанной на бумаге программы в память ПВМ — ввести ее с клавиатуры. В будущем, возможно, для считывания напечатанного на бумаге текста и ввода его в ПВМ появятся специальные оптические читающие устройства, но пока подобных устройств нет.

Чтобы иметь уверенность в работоспособности программы, введенной с готового напечатанного текста, нельзя допускать ни малейшего отклонения от оригинала при наборе на клавиатуре. Эта операция требует большой точности, хотя она не слишком утомительна, если программа достаточно мала, например содержит не более 300 строк.

Если программа написана не специально для ПВМ, то даже после очень тщательного копирования она может не работать. Это связано с тем, что существуют некоторые различия, как очевидные, так и достаточно тонкие, между версиями Бэйсика для различных вычислительных машин. Никогда нельзя быть заранее уверенным в том, что Бэйсик-программа, написанная для какой-либо другой вычислительной машины, будет работать на ПВМ; выяснить это можно, лишь пробуя запускать эту программу на ПВМ.

Ввод с клавиатуры в память ПВМ Бэйсик-программы осуществляется с помощью одного из Бэйсик-интерпретаторов. Для большинства программ, написанных не специально для ПВМ, достаточно использовать интерпретатор дискового Бэйсика. Однако если есть некоторые сомнения относительно того, каким интерпретатором

воспользоваться, то следует выбрать интерпретатор расширенной версии, поскольку он допускает более широкий набор команд.

Перед началом ввода программы надо набрать на клавиатуре BASIC или BASICA в зависимости от того, какой требуется интерпретатор, а затем нажать клавишу ↵. После этого управление перейдет к интерпретатору и на экране появится его идентификацион-

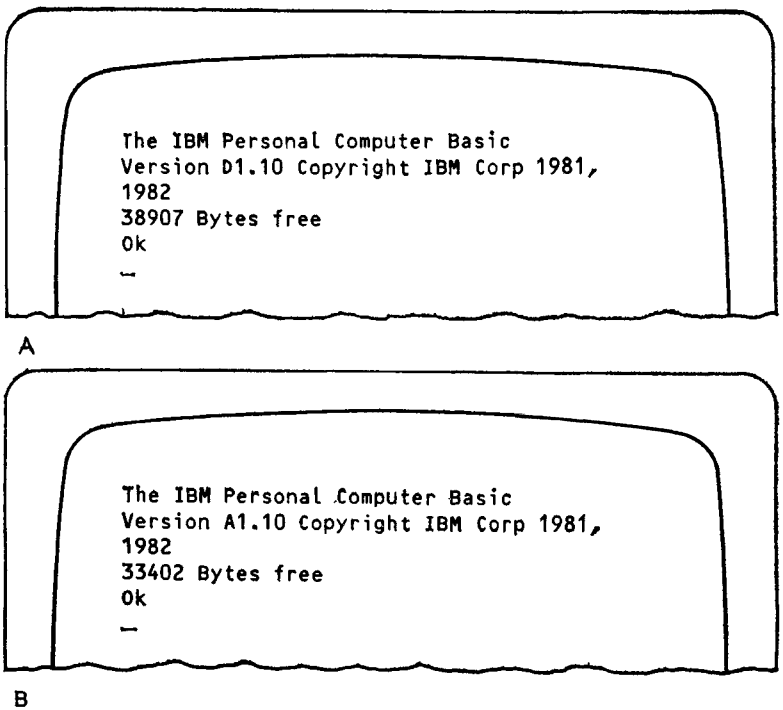


Рис. 4.1. Идентификационные сообщения Бэйсик-интерпретаторов.

А — на экране сообщение, следующего содержания: «Бэйсик ПВМ фирмы IBM. Версия D1.10. Авторское право фирмы IBM 1981. Свободная память 38907 байт»; В — на экране сообщение, следующего содержания: «Бэйсик ПВМ фирмы IBM. Версия A1.10. Авторское право фирмы IBM 1981. Свободная память 33402 байт».

ное сообщение (рис. 4.1). Сообщение занимает 3 строки; на 4-й строке после него высветится "Ok", а на 5-й — курсор.

Заметим, что наводящее сообщение системы ДОС ПВМ здесь уже не появляется, поскольку к данному моменту управление передано от ДОС ПВМ к интерпретатору. Интерпретатор по существу не выводит никаких наводящих сообщений, хотя курсор часто появляется на экране под сообщением "Ok".

После выполнения указанных действий можно начать вводить строки программы по напечатанному оригиналу. При этом разре-

шается использовать любую комбинацию прописных и строчных букв, даже если в напечатанном варианте программы все буквы прописные. Интерпретатор при необходимости автоматически преобразует все буквы в прописные.

Каждая строка программы начинается с некоторого числа, называемого номером строки. Отдельные строки программы могут оказаться слишком длинными и не умещаться на одной строке экрана дисплея. В этом случае следует продолжать набирать строку программы, не обращая внимания на то, что строка экрана уже полностью заполнена: программная строка автоматически перенесется на следующую экранную строку. При наборе таких длинных программных строк не надо нажимать клавишу ↵ каждый раз, когда курсор достигает конца строки экрана. Эту клавишу следует нажимать только после того, как будет набрана вся программная строка, т. е. непосредственно перед началом ввода новой программной строки, начинающейся своим номером.

Пример. Программа упорядочения по алфавиту списка, содержащего до 100 элементов (рис. 4.2).

При вводе текста этой программы с клавиатуры клавишу ↵ надо нажать ровно 33 раза, по одному разу в конце каждой программной строки.

Если при наборе некоторой программной строки перед нажатием клавиши ↵ обнаружилось, что в этой строке была допущена ошибка, то ее можно исправить с помощью клавиши ← или **Esc**. Клавиша ← позволяет, оставаясь на той же строке экрана, вернуться на одну позицию влево, а с помощью клавиши **Esc** можно стереть программную строку.

Выявление ошибок, допущенных при вводе программы с клавиатуры

Поскольку очень важно, чтобы набираемые на клавиатуре строки программы были точными копиями печатного оригинала, перед запуском уже введенной программы следует еще раз проверить все ее строки. Если после ввода программы, когда управление вычислительным процессом все еще продолжает осуществлять интерпретатор (а не ДОС ПВМ), нажать клавишу **F1**, то на экране дисплея появится слово "LIST". После этого можно нажать клавишу ↵, и тогда на экране начнут высвечиваться введенные ранее строки программы. Одновременно на экране дисплея помещаются 22 строки. Если программа содержит большее число строк, то каждый раз при заполнении всего экрана верхняя строка будет исчезать, все остальные сдвигаться вверх, а освободившуюся нижнюю строку экрана займет новая программная строка. В результате по экрану быстро пробегут все строки программы и останутся лишь 22 последние. В такой ситуации следует снова нажать клавишу **F1**, но теперь до

```
10 DIM ITEM$(100)
20 CLS
30 PRINT ,"ALPHABETIZE"
40 PRINT
50 INPUT "Введите количество элементов"; NBR
60 FOR J=1 TO NBR
70 PRINT "Введите очередной элемент"; J
80 INPUT ITEM$(J)
90 NEXT J
100 N=NBR
110 N=INT(N/2)
120 IF N=0 THEN 240
130 J=1
140 K=NBR-N
150 L=J
160 M=L+N
170 IF ITEM$(L)<=ITEM$(M) THEN 210
180 SWAP ITEM$(L),ITEM$(M)
190 L=L-N
200 IF L>=1 THEN 160
210 J=J+1
220 IF J>K THEN 110
230 GOTO 150
240 CLS
250 FOR J=1 TO NBR
260 PRINT ITEM$(J)
270 NEXT J
280 INPUT "Продолжать печать (Y/N)";A$
290 IF LEFT$(A$,1)="Y" OR LEFT$(A$,1)="y" THEN 240
300 END
```

Рис. 4.2. Программа упорядочения по алфавиту.

нажатия клавиши ↵ нужно подготовиться к тому, чтобы сразу после первого заполнения всего экрана быстро нажать одновременно клавиши **Ctrl** и **Num Lock**, тем самым «замораживая» текущее изображение на экране.

Просматривая программные строки, необходимо убедиться в том, что они в точности совпадают с напечатанным оригиналом. Следует удостовериться, что заглавная буква «O» нигде не перепутана с нулем, буква «l» — с единицей, а «Z» — с двойкой. При выявлении типографских ошибок надо записать номер ошибочной строки. После проверки всех поместившихся на экране программных строк можно продолжить просмотр. Для этого вначале следует подготовиться к тому, чтобы в дальнейшем при необходимости снова быстро нажать комбинацию клавиш **Ctrl|Num Lock**. После этого нажимается клавиша ↵ или клавиша пробела и на экране сразу же начинают появляться строки программы; в нужный момент следует быстро нажать комбинацию клавиш **Ctrl|Num Lock**. Далее, начиная с

последней просмотренной в предыдущем цикле строки, надо проверить очередной сегмент программы, записывая номера строк, в которых обнаружатся ошибки. Указанные действия необходимо повторять до тех пор, пока не будет проверена вся программа.

Многие считают, что при проверке правильности введенной с клавиатуры программы удобнее пользоваться ее распечаткой на бумаге. Если в состав системы входит печатающее устройство, то для получения экземпляра распечатки надо нажать клавишу **F1**, а затем **F6**; нажимать клавишу ↵ в данном случае не нужно. После этого на экране дисплея появится сообщение 'LIST, "LPT1:"' и начнется печать текста программы. Печатающее устройство, естественно, предварительно должно быть приведено в состояние готовности к работе.

Вызов программных строк для исправления

Имея список номеров программных строк, в которых были обнаружены ошибки, можно приступить к их исправлению. Вычислительная машина в это время по-прежнему должна работать под управлением интепретатора. Отредактировать можно любую строку программы, выведенную на экран дисплея. Перемещать курсор по экрану дисплея, а также вводить и удалять отдельные символы можно с помощью клавиш малой вспомогательной клавиатуры, если они не используются для набора цифр.

Не имеет значения, каким образом программная строка окажется на экране: как только она там появилась, ее можно изменять. Для того чтобы добраться до нужной строки, можно, например, нажав клавишу **F1**, начать прогон строк программы, а затем при появлении искомой строки остановиться, нажимая **Ctrl|Scroll Lock**. Возможен и другой способ вызвать строку для исправления на экран: надо набрать на клавиатуре команду EDIT, дать пробел, набрать номер требуемой программной строки, а затем нажать клавишу ↵. После этого на следующей строке экрана появится программная строка с указанным номером, а курсор будет находиться под первым ее символом.

Внесение изменений в программную строку

Чтобы исправить ошибку в выведенной на экран программной строке, необходимо сначала передвинуть курсор к местоположению ошибочного символа. Описание клавиш, с помощью которых можно перемещать курсор по экрану, а также некоторых дополнительно используемых при редактировании клавиш приведено в табл. 4.1.

Если курсор находится непосредственно под неправильным символом, то любой набранный в этот момент на клавиатуре символ появится на экране на месте старого, ошибочного (рис. 4.3). Для

Таблица 4.1. **Клавиши редактирования и управления курсором (для работы с Бэйсик-программами)**

Клавиша	Производимое действие
↑	Перемещение курсора на одну строку вверх
↓	Перемещение курсора на одну строку вниз
→	Перемещение курсора на одну позицию вправо
←	Перемещение курсора на одну позицию влево
Ctrl|→	Перемещение курсора на одно слово вправо [1]
Ctrl|←	Перемещение курсора на одно слово влево [1]
Home	Перемещение курсора в верхний левый угол экрана
End	Перемещение курсора в конец программной строки
→|	Перемещение курсора к началу следующей зоны [2]
Esc	Удаление с экрана программной строки
←	Удаление последнего набранного символа
Ctrl|←	Удаление символа, соответствующего текущей позиции курсора
Del	То же, что и для **Ctrl**|←
Ins	Переход к режиму вставки символов в строку или в выход из этого режима [3]
Ctrl|**Home**	Очистка экрана и перемещение курсора в левый верхний угол экрана
Ctrl|**End**	Удаление оставшейся части программной строки
↵	Запоминание в памяти ПВМ всех изменений, внесенных в текущую программную строку

[1] Под «словом» здесь понимается символ или группа символов, начинающаяся с буквы или цифры. Пробелы, знаки препинания и специальные символы отделяют слова друг от друга.

[2] Строка на экране считается разбитой на зоны по 8 символов; первая зона начинается с первой позиции строки, вторая — с 9-й, третья — с 17-й и т. д.

[3] Все остальные приведенные здесь клавиши, кроме →| и ←, также осуществляют выход из режима вставки символов.

удаления символов можно воспользоваться клавишей **Del** или ← (рис. 4.4). Чтобы вставить в строку новые символы, следует нажать клавишу **Ins**, набрать на клавиатуре вставляемые символы и вновь нажать клавишу **Ins** (рис. 4.5).

Перечисленные выше операции изменяют лишь изображение строки на экране. Если после исправления некоторых символов прямо перейти с помощью клавиш ↑ или ↓ к новой строке, то внесенные изменения не введутся в память ПВМ и исправляемая программная строка в действительности не изменится. Для того чтобы все исправления запоминались в ЗУ, необходимо, до того как курсор исчезнет с редактируемой строки, нажать клавишу ↵ (в этот момент курсор, находясь на исправленной строке, может занимать на ней любую позицию).

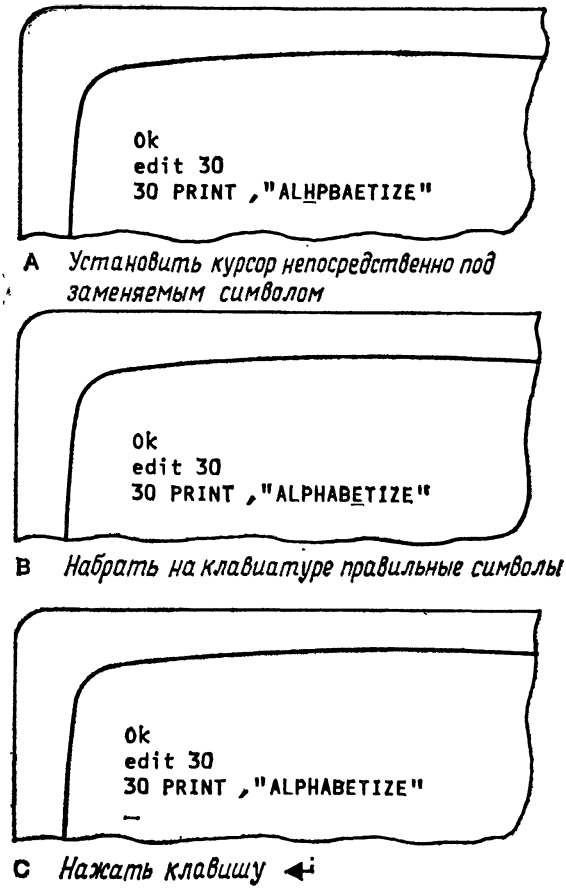
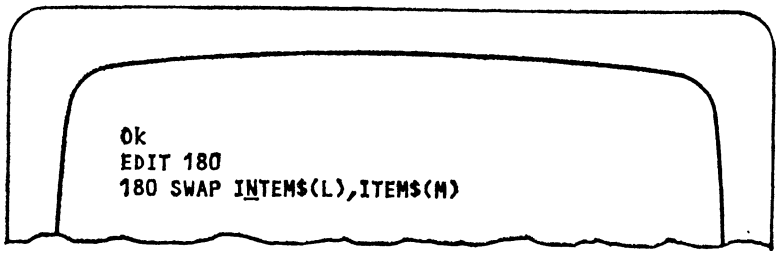


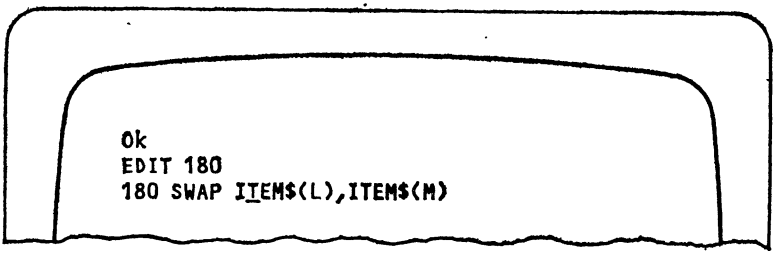
Рис. 4.3. Замена символов в программной строке.

Неправильные номера программных строк

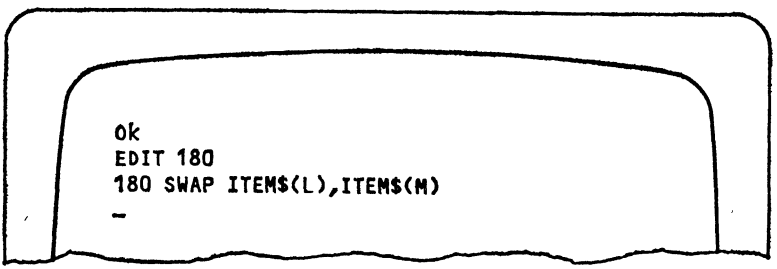
Номера, с которых начинаются все строки Бэйсик-программы, очень важны, поскольку они определяют последовательность выполнения команд программы. Неправильный ввод номеров строк может привести к существенному изменению всей программы, а восстанавливать их в правильном виде довольно сложно. Поэтому надо стараться вводить эти номера правильно с первого раза. Если ошибка все же произошла и программная строка была введена с неправильным номером, то для корректировки необходимо сначала вывести строку на экран и исправить ее номер описанным выше способом, после чего в памяти ПВМ будут находиться две копии одной и той же программной строки: одна с правильным номером, а дру-



A *Установить курсор под удаляемым символом*



B *Для удаления символа нажать клавишу Del*



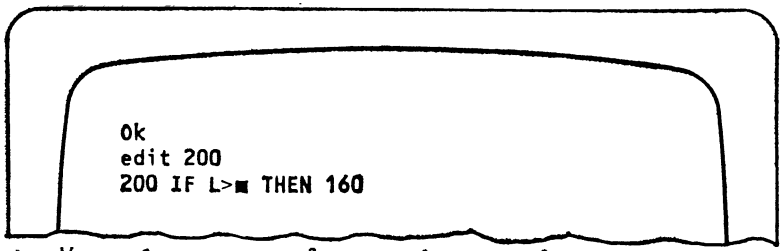
C *Нажать клавишу* ←┘

Рис. 4.4. Удаление символов из программной строки.

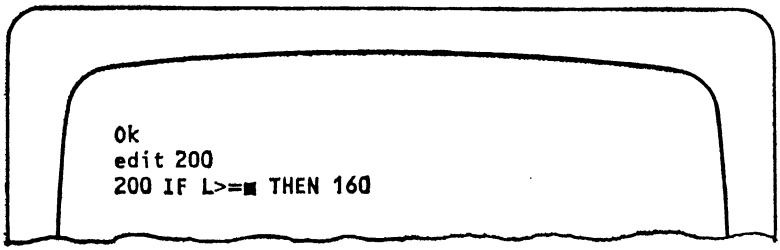
гая с неправильным. Чтобы удалить строку с неправильным номером, достаточно набрать на клавиатуре этот номер, а затем нажать клавишу ←┘.

Запись введенной с клавиатуры программы на диск

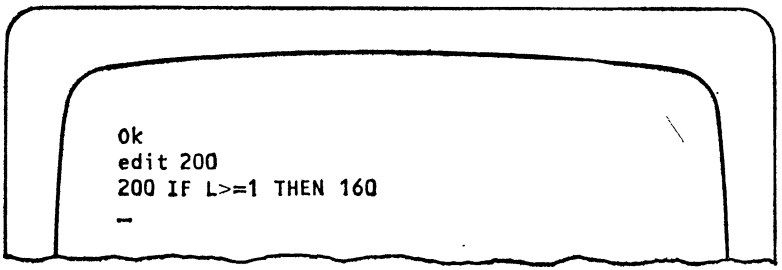
Программа может быть запущена сразу же после того, как она без ошибок введена с клавиатуры, однако для дальнейшего использования целесообразно все же скопировать ее на диск. Программу можно записывать на диск и в процессе ее набора и корректировки и делать это столько раз, сколько потребуется. Такое копирование



A *Установить курсор под тем символом, перед которым нужно вставить новые символы; нажать клавишу Ins*



B *Набрать на клавиатуре вставляемые символы*



C *Нажать клавишу* ←┘

Рис. 4.5. Вставка символов в программную строку.

позволяет разбить работу по вводу и проверке программы на ряд небольших этапов и сохранить результаты уже выполненной работы в том случае, когда во время очередных действий программа, находящаяся в памяти ПВМ, будет случайно стерта.

Поскольку программа записывается на диск в виде файла, прежде всего необходимо выбрать имя для этого файла. При формировании имени следует руководствоваться правилами образования имен файлов, описанными в гл. 3 (рис. 3.6). При этом можно не беспокоиться о том, какое взять окончание, поскольку система автоматически ставит в конце любого такого имени файла символы .BAS.

После определения имени файла надо установить правильно раз-

меченный дискет на имеющийся дисковод или выбрать винчестерский диск, на котором есть свободное место. Затем следует нажать клавишу **F4**, и на экране появится сообщение «SAVE». Далее необходимо набрать на клавиатуре идентификатор дисковода, выбранное имя программного файла и нажать клавишу ↵. В течение нескольких секунд будет происходить активация дисковода, и на это время курсор исчезает с экрана. Затем он снова появится вместе с сообщением "Ok". Если произошел сбой и запись программы на диск не завершилась, то на экран будет выдано соответствующее сообщение об ошибке. В этом случае следует попробовать повторить запись программы, используя другой дискет, другой дисковод или новое имя файла.

Загрузка программы с диска в память ПВМ

Записанную на диске программу можно переслать в память ПВМ без выполнения. При этом загружаемая программа вытеснит из памяти программу, находившуюся там до момента загрузки. Поэтому, если текущую программу требуется сохранить, надо внимательно следить за тем, чтобы перед загрузкой новой программы старая была скопирована на диск или другой носитель.

Прежде всего следует проверить, установлен ли нужный дискет на дисководе, воспользоваться им, а затем набрать на клавиатуре идентификатор дисковода, имя программного файла и нажать клавишу ↵. На несколько секунд, пока дисковод активизируется, курсор исчезнет с экрана. Когда он появится вновь вместе с сообщением «Ok», это будет означать, что пересылка программы с диска завершена. Если почему-либо программа не загрузилась, на экран будет выдано еще и сообщение об ошибке. В этом случае следует попытаться повторить загрузку, внимательно проверяя правильность набора и используя другой дисковод или другую копию дискета.

Запуск введенной с клавиатуры программы

После ввода программы с клавиатуры или загрузки с диска ее можно запустить на выполнение с помощью клавиши F2. После нажатия этой клавиши на экране появится слово «RUN», а за ним — первое сообщение запускаемой прикладной программы. Можно поступить иначе: записать программу на дискет, передать управление системе ДОС ПВМ, а затем запускать программу как любую хранящуюся на дискете прикладную программу. Для возврата управления системе ДОС ПВМ следует ввести с клавиатуры команду SYSTEM.

Пример

```
Ok
system
A>
```

Глава 5

ДОПОЛНИТЕЛЬНЫЕ СРЕДСТВА ДОС ПВМ

В данной главе рассматриваются различные модификации ранее описанных команд, расширенные средства редактирования, а также возможные способы комбинирования существующих системных команд для создания собственных команд пользователя и методы организации автоматического выполнения любой последовательности команд после запуска ДОС ПВМ [1].

Соединение файлов

Несколько текстовых файлов могут быть объединены в один с помощью команды COPY. Для этого задают в явном виде имена двух или более исходных файлов, отделяя их друг от друга знаком плюс, и имя результирующего файла.

Пример

```
A>copy ch1.txt+ch2.txt+ch3.txt b:book.txt
```

В результирующий файл сначала копируется содержимое первого исходного файла, непосредственно за ним — содержимое второго, затем — третьего и т. д. до тех пор, пока не будут скопированы все исходные файлы. По завершении копирования на экране появляется сообщение "1 File(s) copied" («Создано файлов-копий 1»).

Идентификаторы дисководов как перед именами исходных файлов, так и перед именем результирующего указывать не обязательно. Если идентификатор дисковода в явном виде не задан, то выбирается дисковод, принятый по умолчанию. Можно также опустить и имя результирующего файла. Тогда в качестве результирующего файла используется первый исходный файл и к нему присоединяются все остальные исходные файлы.

Родовые имена файлов

Имена исходных и результирующего файлов, используемые в команде COPY, могут быть родовыми.

Пример. Соединение всех файлов дисковода B с именами, имеющими расширение TXT, в один файл.

Результат этой операции записывается в файл BOOK.DOC, находящийся на носителе A:

```
A>copy b:*.txt book.doc
```

[1] В главе содержатся сведения, которые могут представлять интерес даже для пользователей, которые предполагают при работе с ПВМ использовать только готовые пакеты программного обеспечения.

После завершения копирования на экран дисплея выводится число созданных результирующих файлов.

Используя родовые имена, можно с помощью всего лишь одной команды COPY присоединить один файл сразу к нескольким другим.

Пример

```
A>copy *.doc+advert.txt b:*.txt
```

В этом примере к каждому файлу носителя A, имеющему расширение имени ДОС, присоединяется файл ADVERT.TXT, и каждая полученная в результате комбинация копируется на носитель B в файл с расширением имени TXT.

Если на диске достаточно свободного места, то можно объединить в один все файлы одного и того же типа.

Пример

```
A>copy all.txt+*.txt
```

Здесь все файлы, имена которых оканчиваются на TXT, присоединяются к файлу ALL.TXT. Для выполнения подобной команды необходимо, чтобы первый указанный в ней файл (в данном случае файл с именем ALL.TXT) уже существовал.

Однако родовые имена файлов не всегда можно использовать при объединении. Так, например, в случае команды COPY не допускается, чтобы заданное в явном виде имя результирующего файла встречалось среди списка имен исходных файлов. Вместе с тем довольно трудно предугадать, в какой момент такая ошибка будет выявлена.

Пример

```
A>copy *.txt all.txt
```

По этой команде сначала предпринимается попытка произвести те же действия, что и в предыдущем примере. Но если файл ALL.TXT уже существовал, то в процессе выполнения команды произойдет ошибка. Ошибка возникнет в тот момент, когда дойдет очередь до копирования файла ALL.TXT (самого в себя), поскольку к этому времени его содержимое могло измениться (с момента начала выполнения команды). На экране появится сообщение "Content of destination lost before copy" («Содержимое копируемого файла потеряно»), но действие команды на этом не завершится и продолжится копирование следующего файла. Полученный в результате файл ALL.TXT будет содержать данные всех TXT-файлов, кроме своего собственного содержимого в исходном состоянии, т. е. до выполнения объединения.

Ошибки могут возникать и тогда, когда родовые имена присвоены и нескольким исходным файлам, и результирующему файлу. Несмотря на то что в руководстве фирмы IBM по дисковой операционной

системе *"Disk Operating System", 2nd Ed.* утверждается, что такая комбинация допустима и не приводит к ошибкам, эксперименты свидетельствуют об обратном. Поэтому при наличии родовых имен у нескольких исходных файлов и у результирующего файла необходимо следить за тем, чтобы перед выполнением объединения все участвующие в этой операции файлы были скопированы. После выполнения такого объединения следует проверить правильность полученного файла результатов.

Вывод содержимого файла на экран дисплея

Содержимое любого файла можно вывести на экран с помощью команды TYPE.

Пример

```
A>type b:address.dat
```

Если файл находится на дисководе, принятом по умолчанию, то идентификатор дисковода перед именем файла можно не указывать. С помощью клавиши **PrtSc**, используемой совместно с клавишей **Ctrl** или ⇧, можно выдать высвеченные на экране дисплея данные на печатающее устройство (гл. 4).

Указанный выше способ вывода данных применим только в случае файлов, содержащих текстовую информацию, как, например, при символьной обработке, при работе с некоторыми файлами данных или с фрагментами программных файлов. Это связано с тем, что обычно программные команды хранятся на диске в сжатом виде: каждая из них представлена специальным одно- или двухбайтовым кодом. При выполнении команды TYPE эти командные байты декодируются так, как если бы они соответствовали обычным отдельным символам, вследствие чего на экране появляются бессмысленные сочетания.

Сравнение файлов

Сравнение содержимого файлов осуществляется с помощью команды COMP. Файлы при этом могут размещаться как на одном и том же, так и на разных дисководах. Для выполнения сравнения сначала следует набрать на клавиатуре командное слово COMP и имена обоих сравниваемых файлов.

Пример

```
A>comp b:program.bas b:program.bak
```

Если файл находится не на принятом по умолчанию дисководе, то его имени обязательно должен предшествовать идентификатор соответствующего дисковода. После набора указанной строки нуж-

но нажать клавишу ↵, в результате чего на экране дисплея появится сообщение с требованием подготовить дискеты:

```
Insert diskette(s) with files to compare
and strike any key when ready

(Установите дискет(ы) со сравниваемыми файлами
и по готовности нажмите любую клавишу)
```

При выполнении команды сначала сравниваются размеры файлов. Если они не совпадают, то операция сравнения на этом завершается. В противном случае содержимое двух файлов сравнивается байт за байтом (посимвольно). При успешном завершении сравнения выдается одно или более сообщений. В случае выявления несоответствия выводятся данные о местоположении несовпадающих байтов (относительно начала файла) и сами несовпадающие значения. Заметим, что не столько важны эти конкретные числа, сколько сам факт обнаружения ошибки. После выявления более десяти несовпадений сравнение прекращается.

Пример. Протокол сравнения двух файлов.

```
Compare error at offset E
File 1 = 64
File 2 = 67
Compare error at offset 13
File 1 = 6B
File 2 = 65
Eof mark not found
Compare more files (Y/N)?
(Ошибка сравнения при смещении E
Файл 1 = 64
Файл 2 = 67
Ошибка сравнения при смещении 13
Файл 1 = 6B
Файл 2 = 65
Не обнаружен признак конца файла
Сравнивать следующие файлы (Да/Нет)?)
```

Сообщение "Eof mark not found" («Не обнаружен признак конца файла») не обязательно указывает на ошибку. Оно является вполне обычным сообщением при сравнении программных файлов, а также нескольких файлов данных. Сообщение "Files compare ok" («Успешное сравнение файлов») означает, что по содержанию файлы идентичны.

Если сравниваемые файлы имеют одинаковые имена, то они должны находиться на разных дисководах, и тогда имя второго файла указывать не обязательно, достаточно задать лишь идентификатор соответствующего этому файлу дисковода;

если же имена файлов различны, то они должны быть указаны в явном виде для обоих файлов. В тех случаях, когда файл находится не на дисководе, принятом по умолчанию, перед его именем должен стоять идентификатор дисковода.

В команде COMP допускается использование родовых имен файлов, но в этом случае в сравнении будет участвовать только первый файл из всех, имеющих заданное родовое имя. Другими словами, одна команда COMP может обеспечить сравнение содержимого только двух файлов.

Установка даты и времени

Электронный часовой механизм, встроенный в ПВМ, фиксирует время в часах, минутах, секундах и сотых долях секунды. Этот механизм связан с механизмом смены даты. При невыключенной вычислительной машине часы работают непрерывно и в полночь происходит смена числа, а при необходимости месяца и года. При перезагрузке ДОС ПВМ имеет место сброс счетчика времени и даты, и в этом случае необходимо ввести их новые значения, которые затем будут использоваться в дисковых справочниках.

При необходимости в любой момент можно проверить текущие и установить новые значения даты и времени с помощью команд DATE и TIME. Для этого достаточно просто набрать на клавиатуре соответствующее командное слово (DATE или TIME) и нажать клавишу ↵: на экране дисплея появятся текущее значение даты или времени и запрос их нового значения. Если нет необходимости изменять текущее значение даты или времени, то в ответ на этот запрос можно просто нажать клавишу ↵, не вводя никаких данных.

Чтобы изменить дату, следует ввести новое название месяца, дня недели и год, разделяя их дефисом или косой чертой. Для изменения времени надо ввести новые значения часов, минут и секунд, отделяя их друг от друга двоеточием. Секунды можно задавать как в виде целых, так и в виде дробных чисел. При задании времени разрешается опускать секунды или даже минуты и секунды, вводя лишь новое значение часа. В этом случае опущенные величины устанавливаются равными нулю.

Пример. Использование команд DATE и TIME.

```
A>date
Current date is Sun 4-01-1984
Enter new date: 4/2/84

A>time
Current time is 10:43:38.70
Enter new time: 17:15
A>_
```

Во второй и пятой строках стоят текущие значения даты и времени соответственно, а в третьей и шестой — запросы новых значений даты и времени, а также ответы на эти запросы.

Копирование программ ДОС ПВМ

Не исключено, что для использования приобретенного пакета прикладных программ потребуется система ДОС ПВМ, непредусмотренная в составе этого пакета. Тогда с помощью команды SYS все программы ДОС ПВМ из памяти машины можно скопировать на любой заданный дисковод.

Пример

```
A>sys b:
```

Идентификатор дисковода можно не указывать, если он соответствует дисководу, принятому по умолчанию. Команда SYS применима только для дискетов, размеченных с использованием опции "/S". Если же дискет не удовлетворяет этому условию, на экране дисплея появляется сообщение "No room for system on destination disk" («На указанном диске нет места для операционной системы») и копирование ДОС ПВМ не производится.

Редактирование команд ДОС ПВМ

Несмотря на то что для ДОС ПВМ выполняется большинство описанных в гл. 2 соглашений, касающихся клавиатуры, имеются клавиши, которые в этой системе действуют не стандартным образом, а в соответствии с новыми определениями. Такими клавишами являются группа клавиш с подсветкой, расположенных на клавиатуре справа, и клавиши малой двухрежимной клавиатуры в нецифровом режиме. ДОС ПВМ связывает также специальные определения со всеми функциональными клавишами. Эти новые определения облегчают вввод, изменение и повторный ввод команд ДОС ПВМ.

Высвечиваемые и хранимые команды

После нажатия клавиши ↵, указывающей на завершение ввода команды ДОС ПВМ, производятся два действия:

- во-первых, ДОС ПВМ выполняет команду, высвеченную на экране дисплея;
- во-вторых, под управлением ДОС ПВМ эта команда копируется с экрана в специально отведенный для этого участок динамической памяти. Эта копия вводимой команды называется *хранимой командой* или *командным шаблоном*.

Указатель хранимой команды

В ДОС ПВМ для хранимых команд предусмотрен специальный символьный указатель, играющий роль второго, невидимого, курсора. В процессе набора команды на клавиатуре истинный курсор перемещается по экрану дисплея вперед вдоль строки. В это же самое время указатель продвигается вперед по символам хранимой команды. Указатель и курсор можно перемещать независимо друг от друга с помощью некоторых переопределенных клавиш. Кроме того, хранимую команду можно вызывать на экран дисплея для редактирования и повторного выполнения.

Малая цифровая клавиатура

В ДОС ПВМ цифровая двухрежимная малая клавиатура не используется для управления курсором. Поэтому, если нажать клавишу **Num Lock**, переводя малую клавиатуру (табл. 5.1) в нецифровой режим, а затем нажать любую из клавиш **Home, End**, ↑, ↓,→ или ←, то не следует ожидать привычного перемещения курсора

Таблица 5.1. **Результаты операций на малой цифровой клавиатуре (в нецифровом режиме) в системе ДОС ПВМ**

Клавиша	Воздействие на высвеченную на экране команду	Воздействие на хранимую команду
←	Возврат на одну позицию влево — удаление крайнего левого символа	Перемещение указателя на один символ влево
→	Вывод на экран следующего символа хранимой команды	Перемещение указателя на один символ вправо
Ins [1]	Никакого воздействия; все последующие набираемые на клавиатуре символы появляются на экране независимо от того, находится ли система в режиме вставок или выведена из него	Перевод системы в режим вставок: при последующем наборе на клавиатуре символов указатель не перемещается. Вывод из режима вставок: при последующем наборе на клавиатуре символов происходит перемещение указателя
Del	Никакого воздействия	Перемещение указателя на один символ вправо
Все остальные клавиши малой клавиатуры	То же	Никакого воздействия

[1]) Если система находится в режиме вставок, то нажатие клавиши **Ins** выводит ее из этого режима. Если система — не в режиме вставок, то нажатие **Ins** переводит ее в этот режим.

по экрану. Клавиши ←, →, **Ins** и **Del** работают при этом в соответствии с некоторыми новыми определениями, а нажатие любой из остальных клавиш малой клавиатуры вообще не вызывает никаких действий.

Функциональные клавиши

В ДОС ПВМ определяются семь из десяти функциональных клавиш (табл. 5.2), расположенных с левого края клавиатуры; остальные три просто не действуют. Покажем, как можно использовать для

Таблица 5.2. **Назначение функциональных клавиш в системе ДОС ПВМ**

Клавиша	Функция
F1	Вывод на экран следующего символа хранимой команды; сдвиг вперед указателя хранимой команды
F2	Ввод следующего набранного на клавиатуре символа; вывод на экран сегмента хранимой команды, оканчивающегося непосредственно перед первым вхождением этого символа (начиная с текущего положения указателя)
F3	Вывод на экран оставшейся части хранимой команды
F4	Ввод следующего набранного на клавиатуре символа; пропуск сегмента хранимой команды, оканчивающегося непосредственно перед первым вхождением этого символа (начиная с текущего положения указателя)
F5	Занесение высвеченной на экране команды в память — создание хранимой команды; сама команда при этом не выполняется
F6	То же самое действие, что и при наборе **Ctrl** | Z
F7	То же самое действие, что и при наборе **Ctrl** | @
F8	Не действует
F9	То же
F10	» »

редактирования клавиши **F1**÷**F5**. Клавиши **F6** и **F7** можно также использовать при работе с программами EDLIN и DEBUG, как это описано в руководстве фирмы IBM по дисковой операционной системе "Disk Operating System".

Повторное использование хранимой команды

Хранимую команду можно вызвать на экран дисплея с помощью клавиши **F1**; при этом с каждым нажатием **F1** на экран выдается один символ.

Пример. Допустим, что заданная часть справочника была выведена только что с помощью команды

```
A>dir b:*.com
```

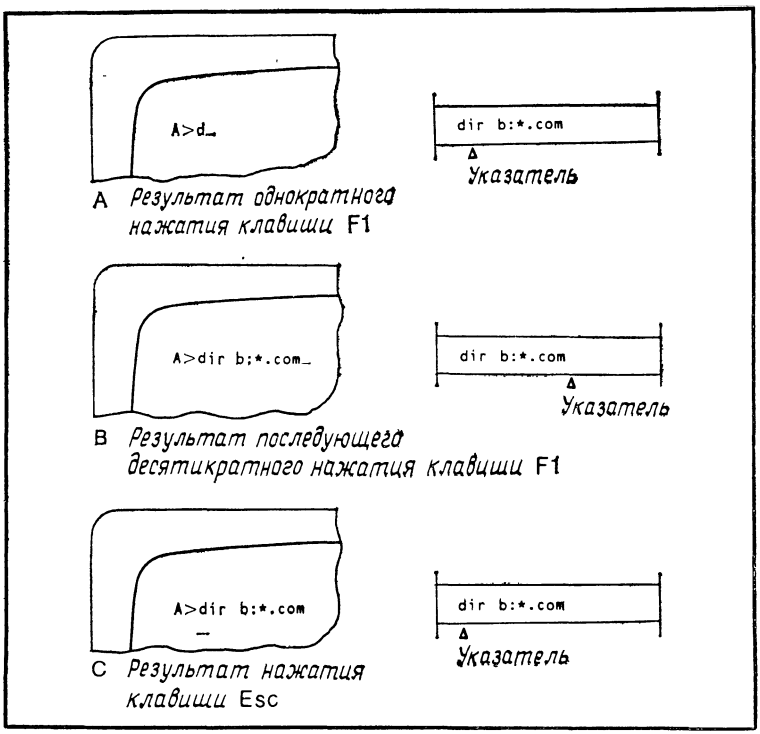


Рис. 5.1. Вывод на экран хранимой команды и удаление выведенной команды с экрана.

Одновременно ее копия — хранимая команда — была занесена в память. Если теперь нажать клавишу **F1**, то на экране появится буква d (рис. 5.1. А). При нажатии клавиши **F1** десять раз подряд на экране вновь появится вся команда (рис. 5.1. В). После этого можно нажать клавишу ←┘, и повторно появившаяся на экране команда выполнится. Для исключения команды можно нажать клавишу **Esc**. В последнем случае команда стирается лишь с экрана дисплея, но не из памяти ПВМ, где она представлена в виде хранимой команды; в памяти изменяется только положение указателя: он снова устанавливается на начало команды (рис. 5.1. С).

В ДОС ПВМ клавиша → действует точно так же, как **F1**, поэтому ее тоже можно использовать для повторного вывода на экран хранимой команды. Каждый раз при нажатии любой из этих клавиш (**F1** или →) на экран дисплея выводится один символ хранимой команды и одновременно указатель хранимой команды сдвигается на одну позицию вперед.

Изменение хранимой команды

Высвеченную на экране команду можно изменить, набирая на клавиатуре различные символы. При этом новые символы занимают место старых, но только в команде, высвеченной на экране; хранимая команда остается без изменения. Как и при нажатии клавиши **F1** или →, при наборе отдельного символа указатель хранимой команды также перемещается вперед.

Пример. Ввод той же команды DIR, что и в предыдущем примере, но уже для дисковода A.

Для ввода команды надо четыре раза нажать клавишу → (или **F1**), в результате чего на экране дисплея появятся четыре первых символа хранимой команды (рис. 5.2. A). Указатель хранимой команды при этом устанавливается на позицию символа B, обозначающего идентификатор дисковода. Затем следует набрать на клавиатуре новый идентификатор дисковода. При выполнении этой операции указатель хранимой команды продвинется на один символ вперед, указывая на двоеточие (рис. 5.2. B). Для вызова на экран оставшейся части хранимой команды можно шесть раз нажать клавишу **F1**. Однако существует более простой способ: достаточно один раз нажать клавишу **F3** и на экран выведется сразу вся оставшаяся часть хранимой команды (рис. 5.2. C). Если затем нажать клавишу ↵, то текущая высвеченная на экране команда занесется в память и станет хранимой командой, а ДОС ПВМ начнет выводить заданный фрагмент справочника дисковода A (рис. 5.2. D).

Пропуск символов хранимой команды

При нажатии клавиши **Del** происходит перемещение указателя хранимой команды вперед без вывода символов команды на экран дисплея. При этом осуществляется своеобразное удаление символов из хранимой команды; в действительности символы не удаляются, а только пропускаются указателем.

Пример. Использование клавиши **Del** для удаления из команды DIR предыдущего примера идентификатора дисковода таким образом, чтобы команда выполнялась применительно к дисководу, принятому по умолчанию.

Для этого сначала надо с помощью клавиш **F1** или → вызвать на экран первые четыре символа хранимой команды (рис. 5.3. A). Затем следует нажать клавишу **Del**, в результате чего указатель хранимой команды сдвинется вперед, а на экране не произойдет никаких изменений (рис. 5.3. B). После этого следует с помощью клавиши **F3** вызвать на экран оставшуюся часть хранимой команды (рис. 5.3. C) и нажать клавишу ↵, если высвеченную на экране команду требуется выполнить, или **Esc**, если эту команду требуется удалить.

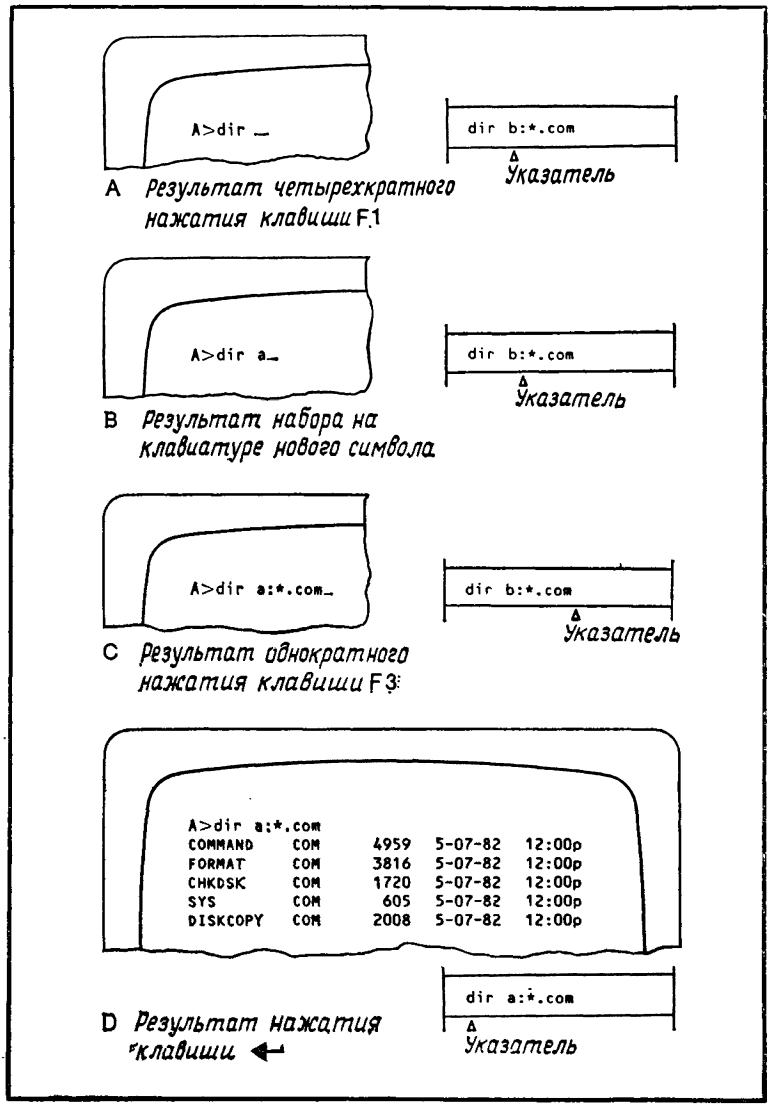


Рис. 5.2. Вывод на экран и изменение хранимой команды.

Вывод на экран фрагментов хранимой команды

Предположим, что после выполнения команды, осуществляющей копирование всех файлов, имена которых имеют расширение COM, с носителя A на носитель B:

A>copy a:*.com b:*.com

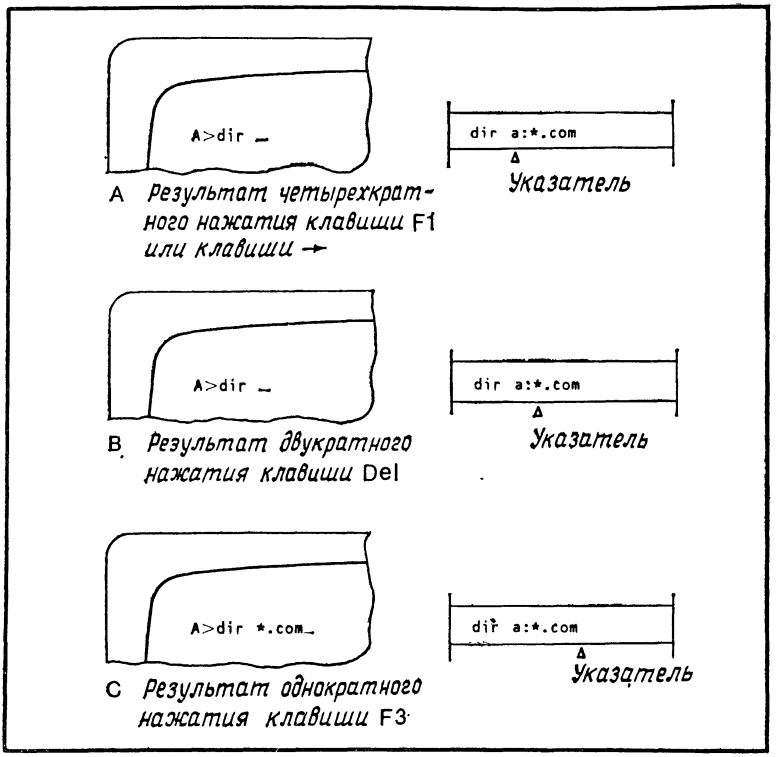


Рис. 5.3. Пропуск символов хранимой команды.

требуется скопировать с дисковода A на дисковод B все файлы, имена которых имеют расширение BAS. Для этого хранимая команда выводится на экран и редактируется с помощью клавиш **F1** и **F3**. Однако при таком способе копирования приходится нажимать клавишу **F1** 14 раз. Частично избежать подобную утомительную операцию позволяет использование клавиши **F2**, так как при этом на экран вводится целый фрагмент хранимой команды.

Для использования клавиши **F2** необходимо определить в хранимой команде ближайший символ, подлежащий изменению. В только что рассмотренной ситуации таким символом является буква «с». Если последовательно нажать клавишу **F2** и клавишу с изображением буквы C, то на экран будут выведены все символы хранимой команды до ближайшей буквы с (рис. 5.4. A). Затем следует набрать на клавиатуре новое расширение имен исходных файлов (рис. 5.4. B) и еще раз нажать клавиши **F2** и **C** для вывода на экран неизменяемого фрагмента хранимой команды до ближайшей буквы с (рис. 5.4. C). После этого остается лишь набрать новое расширение

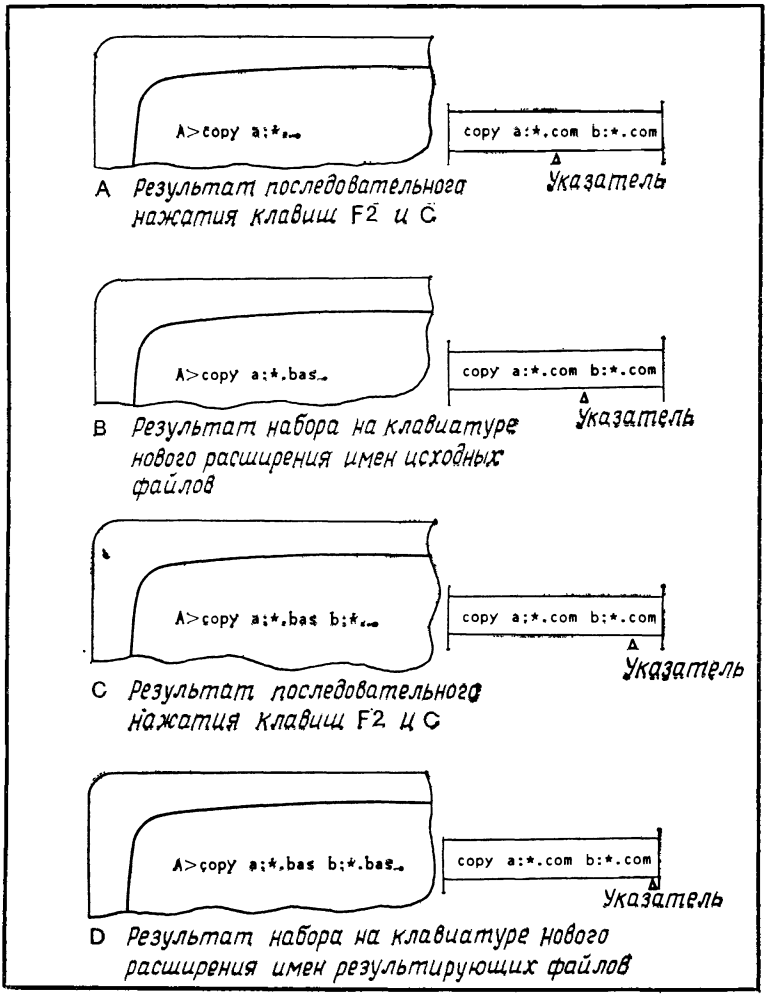


Рис. 5.4. Вывод на экран фрагментов хранимой команды.

для имен результирующих файлов (рис. 5.4. D). Если теперь нажать клавишу ←┘, то выполнится команда COPY, копирующая с дисковода A на дисковод B все файлы, имена которых имеют расширение BAS, а высвеченная на экране команда будет занесена в память и станет хранимой командой.

Отдельный фрагмент хранимой команды, выводимый на экран с помощью клавиши **F2**, начинается с символа, соответствующего текущему положению указателя, и оканчивается непосредственно перед первым вхождением в хранимую команду символа, набранного

сразу после нажатия клавиши **F2**. При этом поиск первого вхождения заданного символа начинается не с символа, соответствующего текущему положению указателя, а со следующего за ним. Если заданный символ не входит в команду (точнее, в ее часть, начинающуюся с текущего положения указателя), то на экран ничего не выводится.

Вставка и замена фрагментов хранимой команды

Любой фрагмент хранимой команды может быть удален и заменен на любое число других символов с помощью клавиш **F4** и **Ins**.

Пример. Сравнение нескольких пар файлов, имена которых имеют одно и то же расширение.

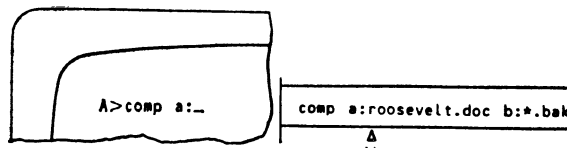
Команда сравнения первой пары файлов может выглядеть так:

```
A>comp a:roosevelt.doc b:*.bak
```

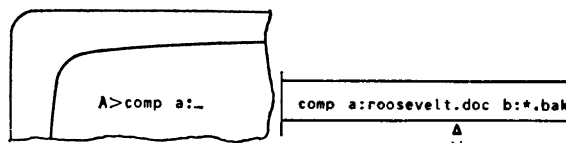
Символ * означает, что при выполнении команды COMP первый файл с заданным в явном виде именем, находящийся на дисководе A, будет сравниваться с файлом дисковода B, имеющим такое же имя, но только с расширением BAK, а не DOC.

Если после выполнения команды сравнения первой пары файлов требуется сравнить файлы TRUMAN.DOC и TRUMAN.BAK, то для этого достаточно заменить в хранимой команде "roosevelt" на "truman", и команда для сравнения новой пары файлов будет готова. Такую замену можно осуществить следующим образом. Сначала с помощью клавиш **F2** и **R** следует вызвать на экран первый фрагмент команды (рис. 5.5. A), чтобы удалить целиком прежнее имя файла. Для этого надо нажать **F4** и набрать на клавиатуре десятичную точку, чтобы в хранимой команде были пропущены все символы, начиная с текущего положения указателя и до ближайшей десятичной точки (рис. 5.5. B). Затем следует нажать клавишу **Ins**, переводя тем самым клавиатуру в режим вставки символов. После этого на клавиатуре должно быть набрано новое имя файла, которое займет соответствующее место в высвечиваемой на экране команде. При этом ни сама хранимая команда, ни ее указатель не изменятся (рис. 5.5.C). Наконец, с помощью клавиши **F3** надо вызвать на экран оставшуюся часть хранимой команды (рис. 5.5.D). Если теперь нажать клавишу ↵, то будет производиться сравнение содержимого новой пары файлов.

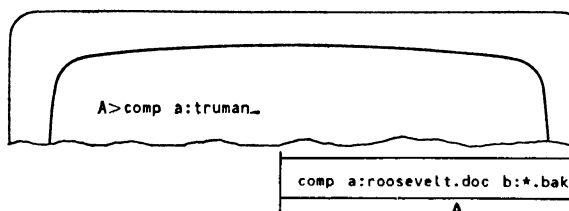
Клавиша **F4** является как бы «усиленной» клавишей **Del**: если с помощью **Del** можно пропустить в хранимой команде одиночный символ, то с помощью **F4** — сразу целый фрагмент хранимой команды. Этот фрагмент начинается с символа, соответствующего текущему положению указателя, и кончается непосредственно перед первым вхождением символа, введенного с клавиатуры сразу после нажатия



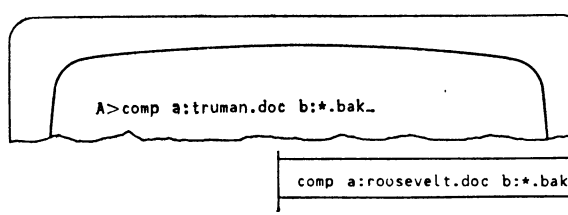
A *Результат последовательного нажатия клавиш* F2 *и* R



B *Результат последовательного нажатия клавиши* F4 *и клавиши* •



C *Результат нажатия клавиши* Ins *с последующим набором на клавиатуре нового имени файла*



D *Результат нажатия клавиши* F3

Рис. 5.5. Замена фрагмента хранимой команды.

клавиши **F4**. При поиске первого вхождения заданного символа просмотр начинается не с символа, соответствующего текущему положению указателя, а со следующего за ним. Если заданный символ не встречается среди символов хранимой команды (начиная с текущего положения указателя), то указатель не сдвигается.

Нажатие клавиши **Ins** переводит систему ДОС ПВМ в режим вставок или выводит ее из этого режима. Между режимом вставок и обычным режимом ДОС ПВМ нет никакой видимой разницы, но форма курсора не может изменяться так, как это имеет место при вводе с клавиатуры Бэйсик-программ. В режиме вставок курсор, как и обычно, продолжает перемещаться вперед по экрану по мере набора на клавиатуре вводимого текста, однако указатель хранимой команды при этом остается на одном и том же месте. Вывод из режима вставок производится с помощью любой из клавиш **Ins, Esc, →, F1, F2, F3, F5** и ←┘.

Запоминание команды без ее выполнения

Текущая высвеченная на экране дисплея команда заносится в память, т. е. становится хранимой в результате нажатия клавиши **F5**, но при этом команда не выполняется. На экране в позиции, соответствующей текущему положению курсора, появляется символ @, указывающий на то, что пересылка команды в память ПВМ завершилась, и курсор перемещается вниз к началу следующей строки экрана. Если на экране не высвечено никакой команды, то при нажатии в этот момент клавиши **F5** производится стирание хранимой команды.

Имена устройств

Некоторые команды ДОС ПВМ позволяют осуществлять обмен информацией не только между дисковыми файлами, но и между другими частями вычислительной системы, к числу которых относятся аппаратные средства последовательного действия, печатающее устройство, клавиатура и экран дисплея. Кроме того, существует так называемое *фиктивное устройство*, которым можно пользоваться при организации тестирования программ. Каждое устройство имеет стандартное имя, используемое точно так же, как и имя любого дискового файла (табл. 5.3).

Наиболее часто имена устройств используются вместе с командой COPY. Например, можно скопировать содержимое дискового файла на экран дисплея с помощью команды, подобной следующей:

```
A > copy  chap3a.doc  con:
```

Заметим, что эта команда производит те же действия, что и команда TYPE.

С помощью команды COPY можно также копировать данные с клавиатуры в дисковый файл. Для этого достаточно в команде COPY в качестве исходного «файла» указать устройство CON: (рис. 5.6.А). Если нажать клавишу ←┘ для выполнения такой команды, то курсор переместится вниз к началу следующей строки экрана. Все, что

Таблица 5.3. **Имена устройств**

Имя устройства [1]	Устройство ввода	Устройство вывода
AUX: [2]	Последовательное устройство	Последовательное устройство
COM1:	То же	То же
CON:	Клавиатура	Экран дисплея
LPT1:	Нет	Основное печатающее устройство
NUL:	Фиктивное устройство для организации тестирования	Фиктивное устройство для организации тестирования
PRN: [2]	Нет	Печатающее устройство

[1] Двоеточие в конце имени устройства является необязательным.
[2] Имена устройств AUX: и PRN: нельзя использовать в команде MODE.

после этого момента набирается на клавиатуре, будет появляться на экране и пересылаться в заданный дисковый файл (рис. 5.6. B). Следует, однако, иметь в виду, что при нажатии клавиш в комбинации **Ctrl|Alt|Del** и **Ctrl|Scroll Lock** никакого копирования не производится, а, как и обычно, происходит прерывание нормального процесса обработки.

Завершить указанную операцию копирования и перевести клавиатуру снова в режим ввода команд можно, набирая на клавиатуре специальную последовательность символов. Эту последовательность легче всего сгенерировать, нажав сначала клавишу **F6**, а затем ←┘, в результате чего на экране сразу появятся символы ^Z (рис. 5.6. C). Следует, однако, иметь в виду, что эти символы приводят к завершению копирования только в том случае, когда они появляются в самом начале экранной строки, т. е. набираются сразу же после нажатия клавиши ←┘. Правильность выполнения операции копирования можно проверить визуально с помощью команды TYPE (рис. 5.6. D).

Если при копировании с устройства CON: нажать клавишу **Esc**, то на экране появится символ \ и уничтожится все, что было набрано вслед за последним нажатием клавиши ←┘. С помощью клавиши ← можно, оставаясь на текущей строке экрана, вернуться на одну позицию влево, но это действие выполняется только для символов, появившихся на экране после последнего нажатия клавиши ←┘.

На клавиатуре нельзя набрать более 127 символов, не нажав при этом хотя бы один раз клавишу ←┘. Если все же попытаться продолжить набор после 127-го символа, то при нажатии клавиши будет выдаваться звуковой сигнал, а на экране не появится ника-

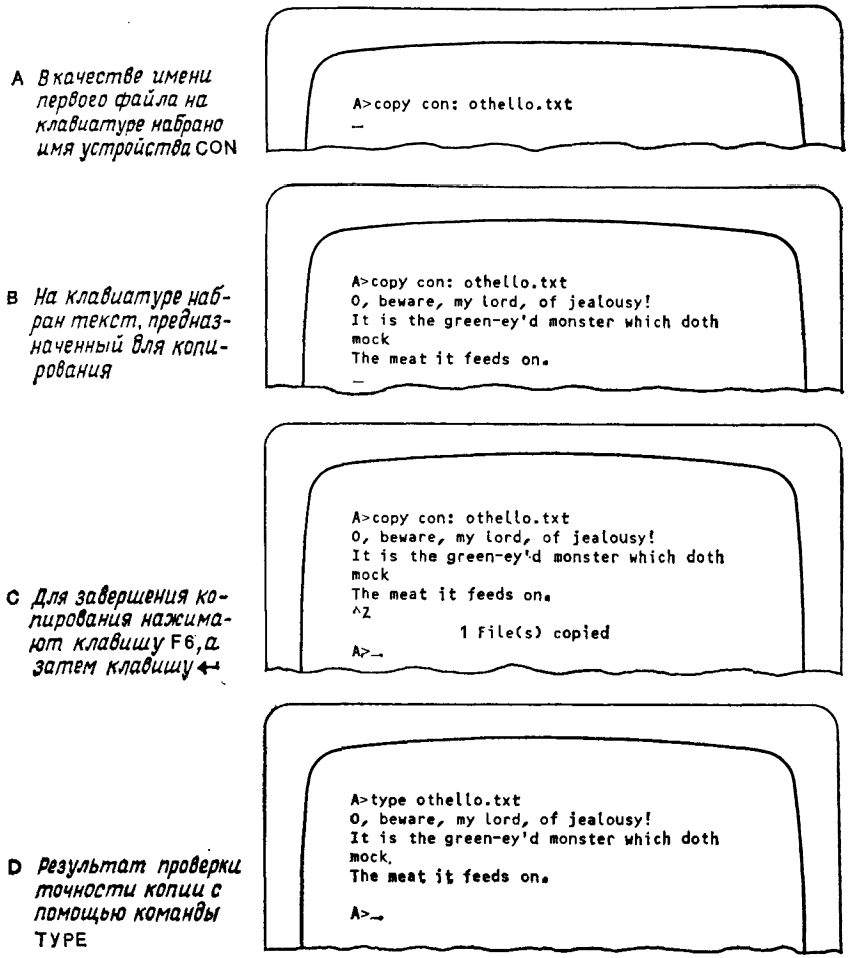


Рис. 5.6. Копирование информации в дисковый файл непосредственно с клавиатуры.

ких новых символов. Для того чтобы продолжить набор, необходимо нажать клавишу ← и начать новую строку экрана.

Набираемые на клавиатуре символы не пересылаются в дисковый файл по одному, а накапливаются в специально отведенной для них области динамической памяти, называемой *буфером*, до тех пор, пока эта область не будет заполнена. Затем вся группа символов пересылается на диск, а в буфере начинают накапливаться очередные набираемые на клавиатуре символы. Если для завершения операции копирования нажать клавишу **F6**, то вся заполненная к

этому моменту часть буфера (начиная с крайней левой позиции) будет переписана на диск в качестве последних заносимых в файл символов.

Пакетная обработка

Как и большинство малых вычислительных машин, ПВМ мгновенно реагирует на каждую введенную в нее команду. Поскольку при этом происходит непрерывное взаимодействие вычислительной машины с человеком, такой способ работы называется *режимом взаимодействия* или *интерактивной обработкой*. Он удобен для задач текстовой обработки, составления небольших коммерческих отчетов или при выполнении персональных вычислений. Однако в ряде случаев удобнее вводить в вычислительную машину сразу целый пакет команд для того, чтобы впоследствии они автоматически выполнялись одна за другой. Такой режим обработки, называемый *пакетным*, предусмотрен в системе ДОС ПВМ.

Создание пакетного файла

Для работы в пакетном режиме прежде всего необходимо создать дисковый файл, содержащий предназначенную для выполнения последовательность команд. Такие файлы называются *пакетными*; имя пакетного файла фактически играет роль новой команды, которую пользователь определяет заранее в виде конкретной последовательности существующих команд.

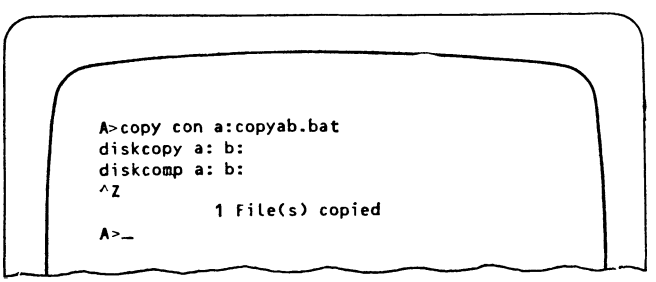


Рис. 5.7. Создание пакетного файла.

Создать пакетный файл можно с помощью команды COPY. Для этого достаточно организовать копирование с клавиатуры (устройство с именем CON:) в пакетный файл. Каждую команду следует набирать так, чтобы она начиналась с новой экранной строки, а для завершения заполнения файла надо сначала нажать клавишу **F6**, а потом ↵. На рис. 5.7 дан пример создания пакетного файла, полезного для многих приложений. Этот файл обеспечивает копи-

рование информации дисковода A на дисковод B с последующим сравнением в целях проверки точности выполненной копии.

Использование команды COPY является самым прямым способом создания пакетного файла. Однако кроме него существуют и другие, например текстовый процессор. (Необходимо иметь в виду, что создаваемый при этом файл должен быть «недокументального» типа, т. е. без какого-либо текстового форматирования.) Можно также использовать предназначенный для ПВМ строковый редактор EDLIN, описанный в руководстве фирмы IBM по дисковой операционной системе "*Disk Operating System*". Для создания пакетного файла можно даже написать специальную программу на Бэйсике или на каком-нибудь другом языке программирования.

Выполнение команд пакетного файла

Для того чтобы начать выполнение последовательности команд, содержащейся в некотором пакетном файле, необходимо набрать на клавиатуре имя этого файла точно так же, как набиралась бы любая команда ДОС ПВМ. В конце имени файла можно дать расширение BAT, хотя это и не обязательно.

Пример. Инициирование пакетного файла COPYAB.BAT (рис. 5.7). Осуществить эту операцию можно, например, так:

```
A > copyab
```

Если после этого нажать клавишу ↵, то на экране появится следующий текст:

```
A > diskcopy  a: b:
Insert source diskette in drive A
Insert target diskette in drive B
Strike any key when ready
_
```

(Во второй строке выведенного текста записано требование на установку исходного дискета на дисковод A, в третьей строке — требование на установку результирующего дискета на дисковод B, а в четвертой — содержится указание нажать по готовности любую клавишу.)

Несмотря на пакетный режим работы ДОС ПВМ, операция копирования фактически начнет выполняться только в том случае, когда пользователь произведет требуемые действия и нажмет какую-нибудь клавишу. После завершения копирования на экране появится сообщение: "Copy another? (Y/N)" («Копировать еще что-либо? (Да/Нет)».) Для продолжения обработки необходимо ввести с клавиатуры ответ на это сообщение, хотя все команды и поступают из пакетного файла. При отрицательном ответе команды пакета продолжают вы-

полняться без последующего ввода с клавиатуры какой-либо дополнительной информации:

```
A > diskcomp a: b:
Insert first diskette in drive A
Insert second diskette in drive B
Strike any key when ready
—
```

(Во второй строке выведено требование установить первый дискет на дисковод А, в третьей строке — требование установить второй дискет на дисковод В, а в четвертой — указание нажать по готовности любую клавишу.)

Пользователь опять-таки должен нажать какую-нибудь клавишу, указывая на то, что дискеты подготовлены к выполнению операции. После завершения сравнения на экране появится сообщение ДОС ПВМ: "Compare more diskettes? (Y/N)" («Сравнивать другие дискеты? (Да/Нет)».) В случае отрицательного ответа пользователя выполнение последовательности команд пакетного файла COPYAB.BAT завершится и на экране появятся символы

```
A >
A > —
```

Переменные пакетного файла

Команды, входящие в пакетный файл, могут содержать фиктивные символы, которые заменяются на действительные с началом пакетной обработки. Эти фиктивные символы называются *переменными* или *параметрами*. Каждая переменная состоит из двух символов, первым из которых всегда является знак процента (%), а вторым — любая цифра от 0 до 9. Таким образом, в каждом пакетном файле может быть до десяти различных переменных: %0, %1, %2 и т. д. Переменные можно использовать в любых командах пакетного файла, а одна и та же переменная в одном и том же пакетном файле может использоваться многократно.

Пример. Пакетный файл для копирования одного файла в другой и сравнения оригинала с копией.

Такой файл можно создать следующим образом:

```
A>copy con copycomp.bat
copy %1 %2
comp %1 %2
^Z
        1 File(s) copied
A> —
```

Ни в одной из команд приведенного выше пакетного файла не задаются в явном виде имена файлов, участвующих в копировании. Фактические имена файлов будут определены при инициализации пакетного файла.

Пример

```
A > copycomp  a:accounts.dat  b:
```

Указанную командную строку можно разделить на три части: в первой части инициируется файл с именем COPYCOMP.BAT; вторая и третья части определяют, на какие фактические символы должны быть заменены переменные %1 и %2 в этом пакетном файле. Если нажать клавишу ↵, то начнут выполняться команды пакета и на экране появятся следующие сообщения:

```
A > copy a:accounts.dat b:
        1 File(s) copied
A > comp a:accounts.dat b:
Insert diskette(s) with files to compare
and strike any key when ready
Files compare ok
Compare more files(Y/N)?
A >
A > —
```

(Во второй строке выведено сообщение об успешном завершении копирования; в четвертой и пятой строках содержатся требование установить дискеты со сравниваемыми файлами и указание нажать по готовности любую клавишу; в шестой строке выдано сообщение об успешном завершении сравнения двух файлов, а в седьмой — запрос: «Нужно ли сравнивать другие файлы? (Да/Нет)». Символы последних двух строк указывают на завершение операции.)

В приведенном выше примере процесса пакетной обработки имена файлов достаточно ввести с клавиатуры только один раз. Заметим, что в процессе обработки пользователю кроме этого придется нажать еще две клавиши: одну — для выполнения операции сравнения файлов, а другую — для завершения этой операции.

Каждой переменной пакетного файла соответствует определенная группа символов в командной строке. Первая группа символов всегда является именем самого пакетного файла (в группу включается и стоящий перед собственно именем файла идентификатор дисковода, если он есть), и она всегда подставляется вместо переменной %0. Вторая группа символов подставляется в пакетном файле вместо переменной %1, и только вместо нее, третья — вместо переменной %2 и т. д.

Для некоторой группы символов может не найтись соответствующей переменной в пакетном файле, и тогда такие символы просто игнорируются. Например, если в пакетном файле нет пере-

менной %0, то заданное в инициирующей его команде имя пакетного файла не будет использоваться в качестве подставляемых символов. Такая ситуация иллюстрируется предыдущим примером.

Вместе с тем может оказаться, что для некоторых переменных пакетного файла в инициирующей командной строке нет соответствующих фактических символов. Тогда при выполнении команд пакетного файла такие переменные игнорируются, что приводит к непредсказуемым последствиям. В одних случаях возникают ошибки, в других пользователю предоставляется возможность вводить недостающие символы в процессе выполнения команд пакета.

Пример

```
A > copycomp b:backup.bat
A > copy b:backup.bat
        1 File(s) copied
A > comp b:backup.bat
Enter 2nd file name or drive id
—
```

(В третьей строке содержится сообщение об успешном завершении копирования, а в пятой — требование на ввод имени второго файла или идентификатора дисковода.)

Здесь в первой инициирующей выполнение пакета команде нет символов, определяющих фактическое имя результирующего файла, т. е. символов, которые можно было бы подставить вместо переменной %2. При выполнении команды COPY в качестве дисковода, соответствующего результирующему файлу, используется принятый по умолчанию дисковод A, и считается, что результирующий файл имеет имя, одинаковое с исходным. Это может, однако, не соответствовать желанию пользователя. В отличие от COPY в команде COMP оба дисковода должны быть заданы в явном виде, поэтому в момент выполнения команды COMP процесс пакетной обработки прерывается и выдается запрос на ввод с клавиатуры недостающего идентификатора дисковода.

Если фактическое имя файла, которое должно использоваться в командах пакета, содержит знак процента, то при задании этого имени символ % надо набрать два раза. Например, для того чтобы внутри пакетного файла использовалось имя файла TAX%.DAT, на клавиатуре следует набрать TAX%%.DAT.

Вывод на экран комментариев

Команда ДОС ПВМ REM предназначена прежде всего для вывода на экран комментария в процессе пакетной обработки. Если первыми тремя символами в командной строке являются символы REM (заглавные или обычные), то на экран выводится вся строка и никаких других действий не производится. Например, добавив к пакетному файлу COPYAB.BAT (рис. 5.7) комментарии, можно

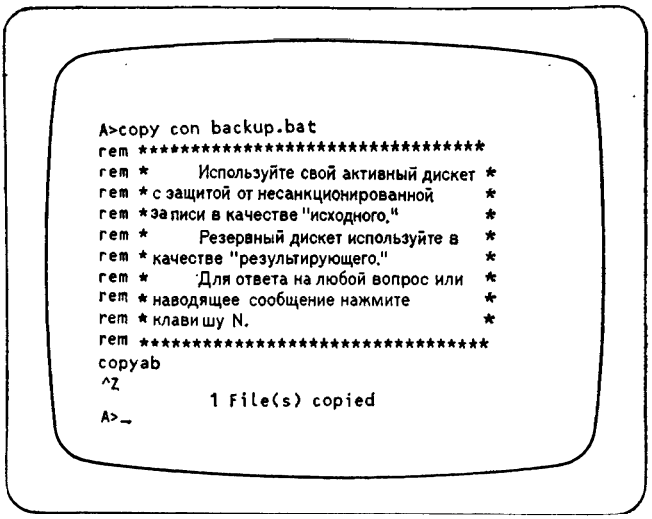


Рис. 5.8. Команды REM в пакетном файле.

превратить его тем самым в общую процедуру копирования. Для этого достаточно создать новый пакетный файл, содержащий ряд команд REM для вывода на экран комментариев и в конце команду, инициирующую выполнение существующего пакетного файла COPYAB.BAT (рис. 5.8).

После командного слова REM должен стоять по крайней мере один пробел; затем следует текст сообщения, которое может быть пустым или содержать до 123 символов.

Организация пауз в ходе пакетной обработки

При выполнении многих команд ДОС ПВМ на экране появляется сообщение типа "Strike any key when ready" («По готовности нажмите любую клавишу»), и процесс выполнения приостанавливается в ожидании от пользователя того или иного действия, в данном случае нажатия клавиши пробела, клавиши ↵ или какой-нибудь другой. С помощью команды PAUSE пользователь может включать в процесс пакетной обработки наряду с указанными стандартными паузами и свои собственные. Например, в пакетном файле COPYAB.BAT используются транзитные команды DISKCOPY и DISKCOMP, и если на дисководе, принятом по умолчанию, нет диска с командными файлами DISKCOPY.COM и DISKCOMP.COM, то ДОС ПВМ не сможет выполнить эти команды. С помощью команды PAUSE на экран можно всегда выводить инструкции для установки нужного диска, и тогда в отсутствие требуемых файлов сообщения об ошибках не появятся (рис. 5.9).

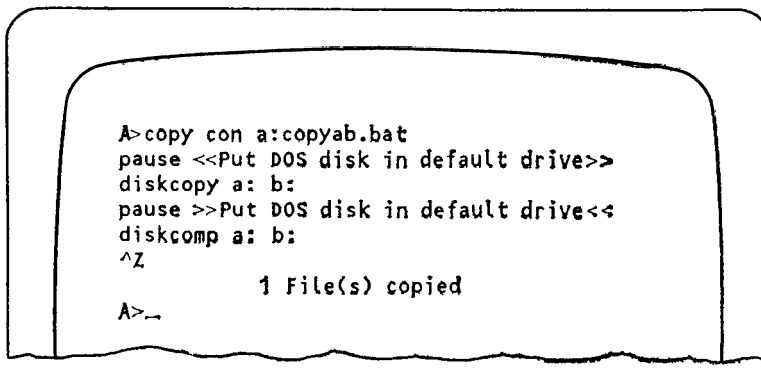


Рис. 5.9. Команды PAUSE в пакетном файле.

Пакетный файл AUTOEXEC

При загрузке ДОС ПВМ путем включения системного блока или нажатия клавиш в комбинации **Ctrl|Alt|Del** система производит поиск на диске А пакетного файла AUTOEXEC.BAT. Если этот файл существует на указанном носителе, то первые команды для ДОС ПВМ поступают из него, а не с клавиатуры. В этом случае система выдает запрос на ввод даты и времени только тогда, когда в файл AUTOEXEC.BAT включены соответствующие команды DATE и TIME. Как и любой пакетный файл, AUTOEXEC.BAT может содержать любое количество допустимых команд ДОС ПВМ, резидентных или транзитных, при этом последние требуют наличия соответствующего командного файла.

Пример. Создание файла AUTOEXEC.BAT, такого, чтобы всякий раз при загрузке ДОС ПВМ выводился запрос даты и времени и производился сдвиг вправо изображения на экране дисплея на 2 символа.

```
A > copy con: autoexec.bat
date
time
mode ,r
mode ,r
^Z
      1 File(s) copied
A > —
```

(В предпоследней строке выведено сообщение об успешном завершении копирования с клавиатуры в файл AUTOEXEC.BAT.)

При загрузке ДОС ПВМ на экране дисплея появятся следующие сообщения:

```
A > date
Current date is Tue 1-01-1980
Enter new date: 2/12/84
A > time
Current time is 0:00:13.29
Enter new time: 16:15

A > mode ,r
A > mode ,r
A >
A > —
```

(Во второй строке выведено сообщение: «Текущая дата — вторник, 1-01-1980», а в третьей содержатся требование на ввод новой даты и ответ на этот запрос (2/12/84). Аналогично в пятой строке выведено текущее время, а в шестой содержится запрос на ввод нового значения времени и ответ на этот запрос.)

При выполнении каждой команды MODE, входящей в пакетный файл, экран очищается и изображение сдвигается на 1 символ вправо; при этом предполагается, что дисплей подсоединен к каналу цветного монитора и графического дисплея.

Часть II

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ БЭЙСИК

Глава 6

ОСНОВНЫЕ ПОНЯТИЯ

В данной и последующих девяти главах рассматриваются вопросы программирования на языке Бэйсик для персональных вычислительных машин, связь Бэйсика с другими языками программирования, описываются простейшие команды вывода информации на экран дисплея, ввода данных с клавиатуры, а также средства редактирования, позволяющие существенно упростить ввод программ в вычислительную машину и исправление ошибок. Показано, как хранить программы на диске и как пользоваться такими программами.

Языки программирования

Чтобы решить ту или иную задачу с помощью вычислительной машины, в последнюю необходимо ввести точные, четко сформулированные инструкции или команды. Составление такой последовательности команд, выполнив которые машина решит поставленную задачу, является целью программирования. Современные вычислительные машины не приспособлены для ввода команд, сформулированных на естественном языке, например английском, испанском, немецком или французском. Поэтому при работе с машиной используют один из специальных языков программирования, главное отличие которых от естественных языков состоит в строго фиксированном грамматическом строе и крайне ограниченном количестве используемых слов.

В основе языка ЭВМ лежит язык ее вычислительного устройства — микропроцессора. Команды такого машинного языка — это последовательности нулей и единиц, которые представляются комбинациями электрических импульсов. Машинный язык непосредственно не используется — работать исключительно только с последовательностями нулей и единиц слишком утомительно. Для общения с машиной служат языки программирования, в какой-то степени напоминающие естественные языки (обычно английский). Программы, написанные на таких языках, транслируются в программы на машинном языке.

Языком программирования, наиболее близким к машинному языку, является *ассемблер*, в котором допускается использование вместо числовых кодов мнемонических имен и отдельных символов.

Однако команды ассемблера больше связаны с внутренней структурой вычислительной машины и протекающими в ней процессами, чем с характером решаемых на этой машине задач. Таким образом, чтобы написать программу на ассемблере, нужно знать, как работает вычислительная машина.

Значительно легче программировать на языках *высокого уровня*, ориентированных не на внутреннее устройство вычислительной машины, а на характер решаемых задач, например на языках Бэйсик, Кобол, Паскаль, Фортран. В любом из них каждая отдельная команда соответствует целой последовательности команд машинного языка, в результате чего программы становятся короче и нагляднее.

Синтаксис языка программирования и составление программ

Любой язык программирования основан на строгих правилах употребления допустимых слов и символов при составлении инструкций. Набор таких правил отдельного языка называется его *синтаксисом*. Одни синтаксические правила достаточно просты и легко запоминаются, другие более сложны и их приходится заучивать наизусть. Так, например, почти во всех языках программирования операция сложения представляется привычным знаком плюс (+), тогда как умножение записывается в виде звездочки (*), а деление в виде косой черты (/), и это объясняется отсутствием на клавиатуре привычных знаков умножения (× или ·) и деления (:).

◆◆◆ Знание набора всех синтаксических правил отнюдь не является достаточным условием для составления программы. Для программирования необходимо еще уметь анализировать задачу, разбивать ее на отдельные логические части, подбирать нужный набор команд, обеспечивающий правильное выполнение каждой части. Навык и умение программировать приобретаются с опытом, причем одним программирование дается легче, другим — труднее.

Бэйсик

Это наиболее популярный язык высокого уровня, используемый для работы на малых вычислительных машинах. Он универсален (на нем можно программировать задачи, связанные с экономикой, промышленностью, научными исследованиями, а также задачи, возникающие в повседневной жизни) и в то же время довольно прост и его легко выучить.

Основы Бэйсика были разработаны в 1964 г. в Дартмутском колледже и оформлены в виде языка, удовлетворявшего скромные потребности пользователей общей большой вычислительной системы. Первоначально при программировании на Бэйсике в качестве

средств передачи данных использовались телетайпы, а не клавиатура и экран дисплея. Кроме того, Бэйсик предоставлял очень ограниченные возможности работы с накопителями на магнитных дисках (НМД). В последующие годы язык развивался и совершенствовался фирмами — производителями ЭВМ: в него были включены средства графического вывода, работы с НМД и другими периферийными устройствами. Из-за отсутствия единого стандарта на все эти расширения языка каждая фирма изменяла язык по-своему. В результате возникло много версий Бэйсика, часто несовместимых. Так, программа, написанная на Бэйсике для вычислительных машин фирм Apple и Radio Shack, могла оказаться непригодной для ПВМ. Однако, зная хотя бы одну версию Бэйсика, легко запомнить особенности любой другой его версии.

Версии Бэйсика ПВМ

Для ПВМ используются три разновидности Бэйсика: *кассетный*, *дисковый* (расширенный кассетный) и *расширенный* Бэйсик (расширенный дисковый). Интерпретатор кассетного Бэйсика всегда хранится в постоянной памяти ПВМ, что обеспечивает возможность трансляции Бэйсик-программ на машинный язык сразу же после включения машины. Интерпретаторы дискового Бэйсика и расширенного Бэйсика необходимо загружать в динамическую память перед началом трансляции программ. Большинство программистов пользуются дисковым Бэйсиком или расширенным Бэйсиком, поскольку кассетный Бэйсик не предусматривает работу с НМД.

Указанные три версии Бэйсика ПВМ имеют много общего, поэтому в дальнейшем будет описываться один язык — Бэйсик ПВМ, конструкции которого входят в любую из трех версий, и только там, где различия между версиями существенны, будут указаны особенности каждой версии.

Начало и окончание работы Бэйсик-интерпретатора

Для ввода и выполнения программы и отдельных команд Бэйсика необходимо передать управление ПВМ одному из трех интерпретаторов. Для каждого интерпретатора эта процедура имеет свои особенности.

Начало работы

Кассетный Бэйсик. Поскольку интерпретатор кассетного Бэйсика всегда находится в постоянной памяти, он готов к работе сразу же, как только подано питание на системный блок. Для запуска интерпретатора кассетного Бэйсика при включенном системном блоке следует открыть крышку дисковода А, а затем привести вы-

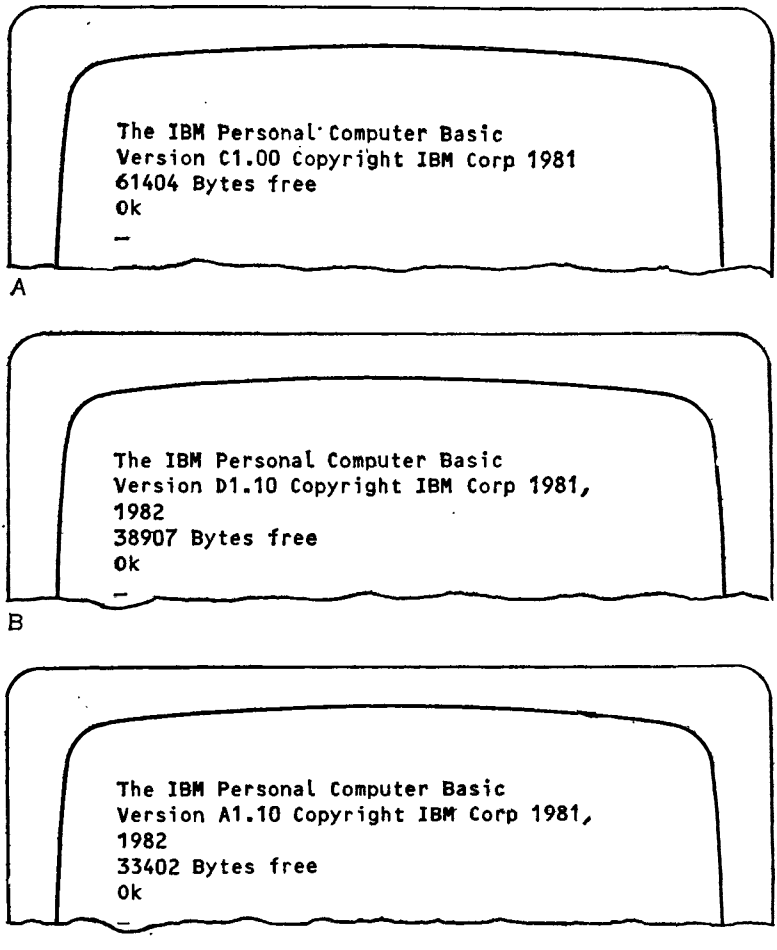


Рис. 6.1. Идентификационные сообщения Бэйсика ПВМ.

А — кассетный Бэйсик. Приведенное на экране сообщение имеет следующее содержание: «Бэйсик ПВМ фирмы IBM. Версия C1.00, 1981 г. Свободная память 61404 байт». В — дисковый Бэйсик. Приведенное на экране сообщение имеет следующее содержание: «Бэйсик ПВМ фирмы IBM. Версия D1.10, 1981—1982 гг. Свободная память 38907 байт». С — расширенный Бэйсик. Приведенное на экране сообщение имеет следующее содержание: «Бэйсик ПВМ фирмы IBM. Версия A1.10, 1981—1982 гг. Свободная память 33402 байт».

числительную машину в исходное состояние нажатием комбинации клавиш **Ctrl|Alt|Del**. В результате экран дисплея очистится и курсор займет исходное положение на экране. В течение 5 с дисковод А будет быстро разгоняться, а затем остановится, поскольку его крышка открыта. Приблизительно через 10 с после этого появится идентификационное сообщение для кассетного Бэйсика (рис. 6.1).

Включение системного блока в отсутствие накопителей на дискетах или при открытых крышках дисководов также приводит к запуску интерпретатора кассетного Бэйсика. При подключении питания выполняется специальный системный тест, и поэтому идентификационное сообщение появляется не так быстро: для этого требуется время от 30 с до 2 мин в зависимости от объема динамической памяти конкретной ПВМ.

Если дисковод А — типа Винчестер, то перед запуском кассетного Бэйсика его необходимо выключить. В тех случаях, когда этого сделать нельзя, вместо кассетного Бэйсика следует использовать дисковый или расширенный Бэйсик.

Дисковый Бэйсик и расширенный Бэйсик. Для запуска как интерпретатора дискового Бэйсика, так и интерпретатора расширенного Бэйсика необходима ДОС ПВМ. Поэтому прежде всего надо произвести загрузку этой системы (гл. 3). Интерпретатор дискового Бэйсика находится в файле BASIC.COM, а расширенного — в файле BASICA.COM. Для запуска любого из этих интерпретаторов следует набрать имя соответствующего файла без конечного слога COM и нажать клавишу ↵. Если нужный файл размещается не на том дисководе, который идентифицируется по умолчанию, то перед именем файла необходимо набрать идентификатор соответствующего ему дисковода.

Пример. Если принятым по умолчанию является дисковод В, а файл с интерпретатором расширенного Бэйсика находится на дисководе А, то запуск производится с помощью следующей команды:

```
B > a:basica
```

Не более чем через 5 с произойдет сброс экрана и появится идентификационное сообщение (рис. 6.1).

Окончание работы

Закончить работу любого из трех интерпретаторов можно путем выключения системного блока. При этом следует соблюдать осторожность, чтобы не отключить питание в момент выполнения какой-либо команды или программы. Для завершения работы интерпретаторов дискового и расширенного Бэйсика можно также использовать команду SYSTEM, которая передает управление операционной системе ДОС ПВМ.

Пример

```
Ok
system
A >—
```

Появление на экране символов A > говорит о том, что вновь произошла передача управления операционной системе ДОС ПВМ. Все Бэйсик-программы, с которыми мы работали до выполнения команды SYSTEM, после ее выполнения удаляются из динамической памяти. Эти программы не восстанавливаются, даже если впоследствии вновь запустить один из Бэйсик-интерпретаторов.

Режимы обработки

Режим немедленной обработки

Сообщение "Ok", появляющееся на экране дисплея в конце идентификационного сообщения, означает, что интерпретатор готов обрабатывать команды Бэйсика. После этого можно работать в так называемом *режиме немедленной обработки* или *прямого общения*, поскольку при таком режиме каждая введенная в машину команда выполняется сразу.

Пример

```
Ok
print "Стремитесь к простоте программ."
```

После набора этой команды на клавиатуре и нажатия клавиши ↵ на экране появится фраза

```
Стремитесь к простоте программ.
Ok
_
```

Таким образом, по указанной команде машина вывела на экран сообщение, которое было заключено в кавычки. Появившееся после этого сообщение Ok (над курсором) свидетельствует о готовности интерпретатора принять следующую команду.

Рассмотренный выше пример показывает, как с помощью оператора PRINT на экране отображается сообщение. Такое сообщение может быть совсем коротким, а может быть и длинным.

Пример

```
Ok
print "Прежде чем начать программировать,
разработайте блок-схему программы".
Прежде чем начать программировать,
разработайте блок-схему программы.
Ok
_
```

В данном примере используется дисплей, ширина экрана которого равна 40 символам. Заметим, что теперь команда PRINT располагается на двух строках; такой автоматический пере-

нос выполняется всегда, когда команда не умещается в одной строке экрана. Нажимать клавишу ↵ следует только по окончании набора всей команды. Отдельный оператор Бэйсика ПВМ может включать в себя до 255 символов: если эта граница нарушается, интерпреватор игнорирует все символы, следующие за 255-м.

Оператор PRINT может также выполнять арифметические действия и полученные результаты выводить на экран.

Пример

```
Ok
print  169/13+87
100
Ok
print 82*14—91
1057
Ok
—
```

Заметим, что ни в одном из операторов данного примера не использовались кавычки, поскольку все, что в операторе PRINT заключено в кавычки, выводится дословно, без изменений. Сравним, например, второй оператор PRINT предыдущего примера со следующим:

```
Ok
print "82*14—91"
82*14—91
Ok
—
```

Сообщения об ошибках

Вводя с клавиатуры ту или иную команду, можно ошибиться и набрать не те буквы или допустить какую-то синтаксическую ошибку. В этом случае команда выполняться не будет и появится сообщение об ошибке.

Пример

```
Ok
prnt 430*22
Syntax error
Ok
—
```

Сообщение "Syntax error" («Синтаксическая ошибка») говорит о том, что интерпретатор не может определить, какую команду следует выполнить, т. е. вводимый текст содержит ошибку. Иногда интерпретатор идентифицирует тип допущенной ошибки. Например,

если в арифметическом выражении вместо числовой переменной или константы встретилось строковое значение, то будет указано, что допущена ошибка, связанная с несоответствием типов данных: на экране появится сообщение «Type mismatch» («Несоответствие типов»).

Пример

```
Ok
print "единица" /2
Type mismatch
Ok
—
```

Не всякая ошибка, допущенная при вводе команды, является синтаксической. Предположим, что надо умножить 555 на 33. Набирая команду, можно случайно вместо знака умножения нажать клавишу со знаком сложения:

```
Ok
Print 555+33
588
Ok
—
```

Хотя введенная команда не дает нужного результата (не вычисляет произведения двух чисел), вид ее вполне допустим и интерпретатор не будет запрашивать повторного ввода.

◆◆◆ В любом случае, если допущена какая-то ошибка, пусть даже не сопровождаемая выдачей сообщения, следует внимательно проверить введенную команду и, выявив ошибку, произвести повторный ввод, но на этот раз более внимательно.

Программируемый режим

Вместо того чтобы вводить команды в режиме немедленного исполнения, можно составить из них программу, которая будет выполняться позже в нужный момент. Такой способ работы с ПВМ называется *программируемым режимом*, *режимом отсроченной обработки* или *режимом непрямого общения*. В режиме отсроченной обработки команды обычно называются *операторами*, хотя часто оба термина употребляются как синонимы. Это связано с тем, что большинство команд режима немедленной обработки можно использовать в качестве операторов режима отсроченной обработки и наоборот. Однако есть такие команды, которые работают только в режиме немедленной обработки, и такие операторы, которые выполняются только в программируемом режиме. Описание всех команд и операторов Бэйсика приведено в приложении А.

Номер строки программы

Характерной чертой программируемого режима является приписывание оператору номера строки, в которой он стоит. Номером строки может быть любое число от 0 до 65529; оно ставится перед оператором.

Пример

```
Ok
10 print "Старайтесь не делать ошибок."
—
```

Выполнение программы

При вводе строки, начинающейся с номера, операторы, содержащиеся в этой строке, сначала помещаются в динамическую память. Поскольку при этом машиной не обрабатывается никакая явная команда, сообщение "Ok" на экране не появляется, хотя система находится в состоянии готовности к приему новой команды или строки программы.

Программа обычно состоит из нескольких строк, каждая из которых имеет свой уникальный номер. Всякий раз, набрав на клавиатуре новую строку, следует нажать клавишу ↵, указывая тем самым, что строка закончена. После этого новая строка поступает в динамическую память и добавляется к уже хранящимся там строкам программы. Для выполнения всей программы вводят команду RUN. Порядок, в котором выполняются строки программы, определяется номерами строк.

Пример. В приводимом ниже фрагменте программы сначала вводятся две строки, которые добавляются к строке предыдущего примера, а затем выполняются все три строки.

```
Ok
20 print "Это требует внимания,
30 print "но экономит время."
run
Старайтесь не делать ошибок.
Это требует внимания,
но экономит время.
Ok
—
```

Команда RUN в ее простейшей форме (в виде одного командного слова) инициирует выполнение программы, начиная со строки с наименьшим номером. Существует, однако, модификация команды RUN, позволяющая запускать программу с любой другой строки: для этого после командного слова RUN указывается номер строки, с которой надо начать выполнение программы.

Пример. Пусть в памяти машины по-прежнему хранятся три строки, введенные в предыдущем примере. Покажем, как можно начать выполнение этой программы со второй строки:

```
Ok
run 20
Это требует внимания,
но экономит время.
Ok
```

—

После выполнения всех строк программы ПВМ вновь переходит в режим немедленной обработки, однако программу можно остановить и раньше с помощью оператора STOP. Если в ходе прогона программы в некоторой строке встретится оператор STOP, то выполнение программы прекращается, выводится сообщение, указывающее точку останова, и система переходит в режим немедленной обработки.

Пример. В приведенную выше программу вначале вставляется промежуточный оператор STOP, и затем осуществляется прогон модифицированной программы.

```
Ok
25 stop
run
Старайтесь не делать ошибок.
Это требует внимания,
Break in 25
Ok
```

—

Сообщение "Break in 25" означает, что программа остановилась на строке с номером 25; ее выполнение можно продолжить с помощью команды CONT.

Пример

```
Ok
cont
но экономит время.
Ok
```

—

Оператор END также прерывает выполнение программы, но в отличие от оператора STOP после выполнения END не выводится сообщение о номере последней выполненной строки. В некоторых случаях можно повторно начать прерванное оператором END выполнение программы с помощью команды CONT.

Прервать выполнение программы можно также одновременным нажатием клавиш **Ctrl** и **Scroll Lock**. При этом, как и при исполь-

зований оператора STOP, на экране появится сообщение о номере последней выполненной строки, а выполнение программы можно будет продолжить с помощью команды CONT.

Ввод операторов языка Бэйсик

Команды с клавиатуры выполняются машиной правильно только при их безошибочном вводе. Ввод операторов, имеющих довольно простой вид и состоящих из одного лишь командного слова, не вызывает никаких трудностей. Однако большинство операторов содержит несколько компонент. При использовании таких операторов необходимо знать правила их применения и следить за тем, чтобы в них присутствовали все обязательные компоненты, в нужном порядке и разделенные соответствующими знаками препинания или должным числом пробелов.

Бэйсик-интерпретатор программ игнорирует все пробелы между номером строки и началом команды. Аналогично не учитывается и большинство остальных пробелов (хотя и не все). В некоторых ситуациях наличие или, наоборот, отсутствие пробела может привести к изменению функций команды.

◆◆◆ При работе с ПВМ используйте примеры, помещенные в этой книге и других источниках, как учебные модели. Будьте особенно внимательны при появлении новых комбинаций операторов, поскольку в этом случае ошибка может проявиться не в момент ввода неверного оператора, а позднее, при прогоне программы, частью которой он является.

При задании командных слов разрешается использовать как прописные, так и строчные буквы в любом сочетании. Все буквы после ввода преобразуются интерпретатором в прописные, но такое преобразование осуществляется не для всех компонент операторов. Так, предыдущие примеры этой главы показывают, что символы, заключенные в кавычки в операторе PRINT, всегда выводятся на экран без изменения.

Управление длиной строки дисплея

Ширина изображений на экране может быть запрограммирована либо на 40 либо на 80 символов в строке с помощью команд WIDTH 40 и WIDTH 80 соответственно. Если в качестве дисплея используется обычный телевизор или цветной видеомонитор, то использовать команду WIDTH 80, как правило, не рекомендуется: изображение символов может оказаться очень нечетким и потому неудобным для чтения.

Длинные команды и строки программы

Несколько следующих друг за другом команд режима немедленной обработки или операторов программируемого режима можно помещать на одной строке. При этом команды или операторы одной строки отделяются друг от друга двоеточием.

Пример

```
Ok
10 print "один" :print "два" :print "три"
—
```

Здесь в одной строке записано три оператора; в общем случае их может быть и больше. Максимально допустимое число символов в одной строке как для программируемого режима, так и для режима немедленной обработки равно 255. Длинная строка может занимать до семи 40-символьных или четырех 80-символьных дисплейных строк. При вводе длинной строки клавишу ↵ нажимают всего лишь один раз после набора последнего оператора или последней команды данной строки.

◆◆◆ Возможно, что изображения выведенных на экран дисплея строк в приведенных примерах будут выглядеть несколько иначе при наборе и вводе тех же самых команд в другую ПВМ. Это связано с тем, что реальный интерпретатор при выводе длинной строки на дисплей неизменно «обрубает» ее на 40-м (или 80-м) символе, даже если перенос приходится на середину некоторого слова (в наших примерах этого не делается).

Использование сокращенных имен команд

Для набора 10 наиболее часто употребляемых команд в Бэйсике ПВМ предусмотрены специальные функциональные клавиши, обеспечивающие ускоренный ввод этих команд. Например, если нажать

Таблица 6.1. **Стандартное назначение функциональных клавиш при программировании на Бэйсике**

Клавиша	Определение	Клавиша	Определение
F1	LIST	F6	,"LPT1:" [1]
F2	RUN [1]	F7	TRON [1]
F3	LOAD"	F8	TROFF [1]
F4	SAVE"	F9	KEY
F5	CONT [1]	F10	SCREEN 0, 0, 0 [1]

[1]) Включая нажатие клавиши ↵.

клавишу **F2**, это вызовет тот же эффект, что и набор команды RUN с последующим нажатием клавиши ↵. При этом в нижней строке экрана высвечиваются команды, соответствующие используемым функциональным клавишам, назначение которых не установлено раз и навсегда. В гл. 11 будет показано, как можно изменять назначение этих клавиш программным способом.

При работе в режиме немедленной обработки действует следующее постоянное соглашение о быстром вводе некоторых команд: одновременное нажатие клавиши **Alt** и одной из 26 буквенных клавиш инициирует ввод заранее оговоренного командного слова, начинающегося с буквы, изображенной на буквенной клавише.

Пример. При нажатии двух клавиш в комбинации **Alt|P** вводится командное слово PRINT. Командные слова, которые можно вводить таким способом, перечислены в табл. 6.2.

Таблица 6.2. **Командные слова, вводимые с помощью клавиши Alt**

Клавиша	Команда	Клавиша	Команда	Клавиша	Команда
A	AUTO	J	J	R	RUN
B	BSAVE	K	KEY	S	SCREEN
C	COLOR	L	LOCATE	T	THEN
D	DELETE	M	MOTOR	U	USING
E	ELSE	N	NEXT	V	VAL
F	FOR	O	OPEN	W	WIDTH
G	GOTO	P	PRINT	X	XOR
H	HEX$	Q	Q	Y	Y
I	INPUT			Z	Z

Для ускорения ввода в большинстве версий Бэйсика, включая Бэйсик ПВМ, разрешается замена командного слова PRINT вопросительным знаком. При обнаружении знака вопроса на месте командного слова интерпретатор всюду подставляет вместо него полное имя PRINT.

Инициирование печатающего устройства

Если одновременно нажать клавиши ⇧ и **PrtSc**, то вся информация, высвеченная в этот момент на экране, будет распечатана на печатающем устройстве точно так же, как в ДОС ПВМ. Однако комбинация клавиш **Ctrl|Prt Sc**, которая также активизировала печатающее устройство в ДОС ПВМ, при работе с Бэйсиком не оказывает на это устройство никакого влияния. Обычно в Бэйсике предусматривается использование параллельного печатающего устройства, но это соглашение можно изменить с помощью команды MODE в ДОС ПВМ (гл. 3).

Комментарии к программе

Иногда можно легко понять, какое действие произведет тот или иной оператор Бэйсика, прочитав соответствующий ему текст. Однако такое возможно далеко не всегда, особенно если речь идет о программе, с которой некоторое время не работали. Для того чтобы понять, что делается в программе, в нее обычно включают операторы REM, содержащие замечания или комментарии, описывающие действия операторов.

Операторы REM не оказывают никакого влияния на ход выполнения программы; единственное их назначение — сделать программу более ясной и понятной. Весь текст строки программы, следующий за командным словом REM, воспринимается интерпретатором как некоторое пояснительное замечание. Поэтому, если комментарий является частью строки, содержащей несколько операторов, соответствующий оператор REM должен быть последним оператором данной строки.

Вместо командного слова REM допустимо ставить просто кавычку; интерпреатор воспринимает кавычку как сокращение для REM. Если комментарий стоит в конце строки, содержащей несколько операторов, а вместо слова REM используется кавычка, то двоеточие перед кавычкой не ставится.

Пример. Использование в программе комментария.

```
Ok
10 rem вывести температуру кипения воды
20 '     для различных температурных шкал
30 print 212 'по Фаренгейту
40 print 100 'по Цельсию
50 print 80:rem по Реомюру
—
```

Редактирование операторов Бэйсика

В Бэйсике ПВМ предусмотрены средства редактирования, с помощью которых можно исправлять либо модифицировать любые команды режима немедленной обработки или строки программ режима отсроченной обработки, пока они видны на экране. Подробно средства редактирования, включая команду EDIT, клавиши редактирования и перемещения курсора, описаны в гл. 4.

Вывод строк программ на экран

После заполнения последней строки экрана изображение сдвигается вверх, и таким образом самые верхние строки исчезают. Как только команда режима немедленной обработки исчезла с экрана, ее дальнейшее редактирование становится невозможным. При

программируемом режиме дело обстоит иначе, так как исчезнувшие строки можно повторно вывести на экран. Для этого предназначена команда LIST, позволяющая выдавать на экран как всю программу, находящуюся в данный момент в памяти машины, так и отдельные ее части.

Пример. Использование команды LIST (предполагается, что в памяти машины хранится программа предыдущего примера).

```
list
10 REM вывести температуру кипения воды
20 '       для различных температурных шкал
30 PRINT 212 'по Фаренгейту
40 PRINT 100 'по Цельсию
50 PRINT 80:REM по Реомюру
Ok
_
```

Заметим, что при повторном выводе программы на экран все командные слова состоят из прописных букв (в данном случае — это слова REM и PRINT).

Если программа занимает в памяти более 22 дисплейных строк, она не умещается на экране полностью. С помощью команды LIST (в простейшем ее виде) можно "прогонять" программу через экран: каждый раз при его заполнении все изображение сдвигается на строку вверх, а освобожденная нижняя строка занимается очередными операторами программы. Для того чтобы в нужный момент остановить такое продвижение, следует одновременно нажать клавиши **Ctrl|Num Lock**. Вывод строк программы можно прервать с помощью комбинации клавиш **Ctrl|Scroll Lock**. Кроме того, в самой команде LIST можно указать диапазон номеров строк, которые нужно вывести.

Пример

```
list 10—20
10 REM вывести температуру кипения воды
20 '       для различных температурных шкал
Ok
List 40 —
40 PRINT 100 'по Цельсию
50 PRINT 80:REM по Реомюру
Ok
list —30
10 REM вывести температуру кипения воды
20 '       для различных температурных шкал
30 PRINT 212 'по Фаренгейту
list 10
```

```
10 REM вывести температуру кипения воды
Ok
```

—

В данном примере показаны четыре варианта команды LIST. В первом из них диапазон выводимых строк задан в явном виде с помощью номеров первой и последней строк. Во втором варианте команды LIST в явном виде задан номер первой выводимой строки; при выполнении такой команды выводятся все строки программы от заданной до последней. При выполнении третьей разновидности команды LIST на экран выдаются строки программы, начиная с первой и кончая строкой с номером, указанным явно в этой команде. С помощью команды четвертого вида выводится одна строка с заданным в ней номером.

В любом варианте оператора LIST вместо явного номера строки можно поставить точку. При выполнении такой команды интерпретатор LIST заменяет точку на номер последней выведенной строки.

Пример

```
list 40
40 PRINT 100 'по Цельсию
Ok
list .—
40 PRINT 100 'по Цельсию
50 PRINT 80:REM по Реомюру
Ok
list .
50 PRINT 80:REM по Реомюру
Ok
```

—

Аналогичным образом строки программы можно выводить на печатающее устройство. Для этого предназначена специальная команда LLIST. Способы задания выводимых строк в ней остаются теми же.

Замена и добавление строк программы

Заменить существующую строку программы на новую очень легко: для этого надо набрать на клавиатуре новую строку с тем же самым номером, что и у старой (заменяемой), и нажать клавишу ↵.

Если между двумя существующими строками Бэйсик-программы требуется вставить новую строку, надо приписать этой новой строке номер в промежутке между номерами смежных с ней строк. После нажатия клавиши ↵ новая строка попадет в память и интерпретатор автоматически поместит ее на нужное место.

Принципиальная трудность возникает лишь в том случае, когда номера строк программы так тесно следуют друг за другом, что невозможно найти число в интервале между номерами двух соседних строк. Во избежание подобной ситуации следует при первоначальном написании программы присваивать последовательным строкам номера, отличающиеся друг от друга на 10. При необходимости можно также заново перенумеровать строки.

Исключение строк и удаление программы из памяти машины

Одну строку программы можно удалить, набрав ее номер и нажав клавишу ←┘. Если же воспользоваться командой DELETE, то можно удалить как одну, так и сразу несколько строк. Формат этой команды похож на формат команды LIST, в чем можно убедиться на приводимых ниже примерах.

```
list 20
20 '      для различных температурных шкал
Ok
delete .
Ok
list 10—30
10 REM вывести температуру кипения воды
30 PRINT 212 'по Фаренгейту
Ok
delete 30—50
Ok
list
10 REM вывести температуру кипения воды
Ok
```

—

Отличие формата команды DELETE от формата команды LIST состоит в следующем. При задании диапазона выводимых на экран строк в команде DELETE необходимо в явном виде указать как номер первой строки диапазона, так и номер последней строки (в случае LIST можно было задавать только один из этих номеров). Таким образом, нельзя исключить все строки программы, начиная с первой и кончая строкой с заданным номером, просто опустив первый номер строки в команде DELETE. Точно так же невозможно удалить и все строки, идущие за первым указанным номером, опустив в команде DELETE второй номер строки, поскольку номера строк, сообщаемые в команде DELETE, должны быть явными номерами существующих строк программы; в противном случае команда DELETE выполняться не будет, и появится сообщение "Illegal function call" («Недопустимый вызов функции»).

Для удаления из памяти сразу всех строк программы предусмотрена команда NEW. Эта операция должна выполняться всякий раз перед вводом очередной программы, иначе ее строки перемешаются со строками старой программы.

Изменение нумерации программных строк

С помощью команды RENUM можно заново перенумеровывать строки. Эта команда позволяет присваивать новые номера одного порядка целому блоку программных строк, начиная со строки с заданным номером и кончая последней строкой, находящейся в памяти. При использовании команды RENUM помимо заменяемого задаются также первый (наименьший) новый номер строки и приращение — разность между двумя соседними новыми номерами.

Пример

```
list
10 REM вывести температуру кипения воды
20 '       для различных температурных шкал
30 PRINT 212 'по Фаренгейту
40 PRINT 100 'по Цельсию
50 PRINT 80:REM по Реомюру
Ok
renum 35,30,5
Ok
list
10 REM вывести температуру кипения воды
20 '       для различных температурных шкал
35 PRINT 212 'по Фаренгейту
40 PRINT 100 'по Цельсию
45 PRINT 80:REM по Реомюру
Ok
```

—

При выполнении команды RENUM в этом примере перенумерация начинается со строки 30, которой дается новый номер 35, а каждой последующей программной строке новый номер присваивается так, что разность между двумя соседними номерами равна 5.

Любой из номеров, задаваемых в команде RENUM, можно опустить. Если при этом отсутствует первый из номеров, то он считается равным 10; если не указан второй номер, то перенумерация начнется с самой первой строки программы. Если не задано третье число, то каждый последующий новый номер будет отличаться от предыдущего на 10.

Номер строки, с которой начинается перенумерация, в команде RENUM должен задаваться так, чтобы строка с указанным номером действительно имелась в программе: в противном случае никакая

перенумерация выполняться не будет, и появится сообщение "Illegal function call" («Недопустимый вызов функции»). Это же сообщение будет выдано и при попытке изменить с помощью команды RENUM последовательность операторов.

Операторы GOTO, GOSUB, IF-THEN, ON-GOTO и ON-GOSUB всегда содержат ссылки на номера различных строк программы; команда RENUM изменяет и эти номера. Если в ходе такой замены встречается ссылка на несуществующую строку, то выдается сообщение, в котором указывается номер отсутствующей строки и номер строки, в которой обнаружена ссылка на нее, а затем процесс перенумерации строк продолжается.

Автоматическая нумерация строк

Для сокращения объема данных, набираемых на клавиатуре при вводе программы, можно использовать средство автоматической нумерации строк — команду AUTO (и тем самым исключить набор номеров строк). В этой команде указывается номер первой строки программы и приращение — число, на которое должны отличаться друг от друга номера двух соседних строк программы.

Пример

```
auto  70,10
70 print 459.67+212 'Ранкин
80 print 273.15+100 'Кельвин
90 —
```

Для прекращения автоматической нумерации строк надо одновременно нажать клавиши **Ctrl** и **Scroll Lock**. При последующем нажатии клавиши **Esc** текущая строка убирается с экрана, но тем не менее ей тоже присваивается номер, который затем приписывается строке, набираемой вместо удаленной.

При автоматической генерации номеров строк может оказаться, что номер очередной строки уже занят находящейся в памяти строкой программы. В этом случае вместе с очередным номером строки на экран выводится кавычка, означающая, что следующая за этой кавычкой строка будет введена на место уже имеющейся в программе строки с тем же номером. (При вводе новой строки в память кавычка заменяется пробелом.) Для сохранения имеющейся программной строки следует, ничего не набирая на клавиатуре, сразу после появления кавычки нажать клавишу ↵. Аналогично с помощью той же клавиши можно пропустить любой автоматически сгенерированный номер, а не только выводимый на экран вместе с кавычкой.

При использовании команды AUTO не обязательно задавать в ней оба числа (номер первой строки и приращение). В зависимости от того, какие числа заданы, а какие — нет, различаются шесть вариантов этой команды (табл. 6.3).

Таблица 6.3. **Допустимые варианты команды AUTO**

Вариант команды	Номер первой строки	Приращение
AUTO	10	10
AUTO 10	Определенный ранее	10
AUTO 10,	» »	Определенное ранее; если приращение ранее не задавалось, то по умолчанию оно принимается равным 10
AUTO, 10	0	Определенное ранее
AUTO,	0	Определенное ранее; если приращение ранее не задавалось, то по умолчанию оно принимается равным 10
AUTO 10,10	Определенный ранее	Определенное ранее

Бэйсик-программы на диске

При работе на ПВМ рекомендуется в каждый момент времени держать в динамической памяти не более одной Бэйсик-программы, поэтому надо иметь возможность хранить программы на устройствах внешней памяти и при необходимости вызывать их оттуда. Для пересылки программ из динамической памяти в дисковые файлы и обратно предназначены пять команд: LIST, SAVE, LOAD, MERGE и RUN. (Кассетный Бэйсик не располагает средствами работы с дисководами, однако с помощью имеющихся в нем команд SAVE и LOAD можно записывать программы в кассетные файлы и считывать их из этих файлов.) Для записи программы на диск или считывания ее с диска необходимо задать имя файла в соответствии с правилами, описанными в гл. 3 (рис. 3.6). Как и обычно, перед именем самого файла может стоять идентификатор соответствующего дисковода. Если такого идентификатора нет, то считается, что файл находится на дисководе, принятом по умолчанию в системе ДОС ПВМ. Можно опускать также и само имя файла: в этом случае интерпретатор будет использовать имя BAS.

Запись программы в дисковый файл

С помощью команд LIST и SAVE осуществляется пересылка Бэйсик-программы из основной памяти в дисковый файл. Если имя файла, в который записывается программа, совпадает с именем файла, уже существующего на указанном дисководе, то старый файл стирается, а на его месте создается новый.

Пример. Две команды, каждая из которых записывает некото-

рую программу в дисковый файл BOILPT.BAS, размещенный на дисководе B.

```
save "b:boilpt"
Ok
list ,"b:boilpt"
Ok
```

—

Команда LIST позволяет пересылать все символы программы на любое внешнее устройство точно так же, как при выводе их на экран дисплея. Команда SAVE отличается от нее тем, что обычно производит сжатие текста программы за счет сокращения длины командных слов до одного символа. Для исключения операции сжатия следует добавить в конце команды SAVE запятую и букву A:

```
save "boilpt" ,a
Ok
```

—

Формат, в котором программа записывается на диск при выполнении команды LIST, похож на соответствующий формат для команды SAVE. (Команду MERGE, описываемую ниже, можно использовать только для программ, записанных в файл с помощью команды SAVE с буквой A в конце.)

Команда SAVE позволяет также записывать программу в файл в зашифрованном виде, защищая ее тем самым от постороннего вмешательства. Если зашифрованная программа снова пересылается в основную память, она становится недоступной для команд LIST и EDIT. При использовании команды SAVE с этой целью следует соблюдать осторожность, поскольку отмена защитных мер становится невозможной. Для того чтобы программа записывалась по команде SAVE в зашифрованном виде, следует поставить в конце этой команды (после заключенного в кавычки имени файла) запятую и букву P.

Пример

```
save "boilpt" ,p
Ok
```

—

При выполнении команды SAVE программа всегда записывается в файл целиком, как и в простейшем случае использования команды LIST. Однако с помощью команды LIST можно записывать в файл и отдельные части программы. Для этого в команде нужно задать соответствующий диапазон изменения номеров строк записываемого фрагмента.

Пример

```
list 10—50, "boilpt"
Ok
—
```

Использование программ, хранящихся в дисковых файлах

Любую программу, помещенную в дисковый файл по команде LIST или SAVE, можно извлечь и вновь разместить в основной памяти с помощью оператора LOAD. При выполнении этого оператора, как и в случае команды NEW, производится очистка памяти, так что новая программа занимает место программы, находившейся в памяти непосредственно перед выполнением команды LOAD.

Пример

```
load "boilpt"
Ok
—
```

С помощью команды LOAD можно не только извлечь программу из внешней памяти, но и выполнить ее. Для этого нужно в конце команды поставить запятую и букву R.

Пример

```
Ok
load "program" ,r
```

Все операции, выполняемые по команде LOAD, можно реализовать, также используя один из вариантов команды RUN: для этого в конце обычной команды RUN надо поставить имя файла, и тогда команда LOAD станет неявной частью команды RUN.

Пример

```
Ok
run "program"
```

Если при выполнении команд LOAD или RUN машине не удается найти указанный файл, то выдается сообщение об ошибке, но программа, находящаяся в этот момент в памяти, не стирается.

Служебные команды для работы с дисковыми файлами

Для удобства пользователя в дисковый и расширенный Бэйсик включены команды, имитирующие три команды ДОС ПВМ. Одна из этих команд предназначена для вывода дисковых справочников, другая — для удаления файлов, а третья — для их переименования.

Команда FILES в ее простейшей модификации позволяет получать полный текст справочника принятого по умолчанию дисковода. С помощью этой же команды можно выводить справочник любого

заданного дисковода и даже получать справочную информацию только об интересующих файлах. В этих случаях нужно задавать в команде FILES идентификатор дисковода, имя файла, а возможно, и то и другое; эти данные указываются в конце команды FILES и должны заключаться в кавычки.

Примеры. Команды FILES (тексты справочников, которые выводятся после выполнения этих команд, опущены).

```
files
files "b:"
files "*.bas"
```

Действие команды FILES больше всего напоминает действие команды ДОС ПВМ DIR/W (гл. 3), хотя между этими командами есть и различия. В обеих командах для задания родового имени файла разрешается использовать символы * и ?, но интерпретация их при выполнении одной команды не совсем такая, как при выполнении другой. Так, в команде FILES требуется в явном виде указывать любое расширение имени файла. Например, по команде FILES "*" будет выдана информация обо всех файлах, имена которых состоят не менее чем из двух символов и не имеют никаких расширений. Для того чтобы получить справочную информацию по всем файлам (с любыми именами), надо использовать команду FILES "*.*". В ДОС ПВМ команды DIR "*" и DIR "*.*" эквивалентны. Еще одно различие между командами FILES и DIR возникает тогда, когда символ * стоит после некоторой буквы, например "P*.*". В этом случае «звездочка» в команде FILES соответствует одному или нескольким символам, а в команде DIR не обязательно, т. е. на ее месте в имени файла может и не быть никакого символа. Например, при выполнении команды FILES "P*.*" не будет выведена информация о файле с именем P.BAS, а при выполнении DIR "P*.*" будет.

Команда KILL позволяет удалить из справочника диска заданное имя файла. При задании в этой команде имени файла не допускается использовать родовые имена файлов, поскольку при выполнении KILL все символы в имени файла интерпретируются буквально. Имя указываемого файла должно быть заключено в кавычки.

Пример

```
kill "temp.bas"
Ok
```

—

С помощью команды NAME-AS можно изменить имя дискового файла. Для этого в ней нужно указать в кавычках одновременно старое и новое имя. Если файл находится не на принимаемом

по умолчанию дисководе, то перед старым именем файла следует поставить идентификатор нужного дисковода. Любой идентификатор, стоящий впереди нового имени файла, игнорируется. Не разрешается использовать обобщенные (родовые) имена файлов.

Пример. Имя файла TEST.BAS заменяется на имя PERM.BAS.

```
name "test.bas" as "perm.bas"
Ok
—
```

Глава 7

КОНСТАНТЫ, ПЕРЕМЕННЫЕ, МАССИВЫ

В данной главе рассматриваются различные типы данных, допустимые в Бэйсике ПВМ, и некоторые способы их использования в программах.

Строковые данные

Строка представляет собой цепочку символов, непосредственно следующих друг за другом. Обычно строковое значение заключается в кавычки:

```
PRINT "улица Мадлен, 25"
```

В строковое значение могут входить прописные и строчные буквы, числа, знаки препинания, т. е. строка может содержать любой из 256 символов, перечисленных в Приложении D. Единственным ограничением является предельно допустимое количество символов: не более 255. Самая короткая строка не содержит никаких символов и называется *пустой* или *нулевой строкой.*

Вычислительная машина работает только с числами и поэтому не может хранить символы в непосредственном виде. При вводе в машину символы преобразуются в соответствии с Американским стандартным кодом для обмена информацией ASCII (American Standard Code for Information Interchange). Например, прописная буква A имеет код 65, а строчная 98. Коды всех допустимых символов приведены в Приложении D.

Большинство символов можно ввести в машину простым нажатием соответствующей клавиши. Однако есть такие символы, которые вводятся специальным способом с помощью клавиши **Alt** и десяти цифровых клавиш малой клавиатуры. Для этого сначала определяют числовой код символа, который нужно ввести. Затем нажимают клавишу **Alt** и, удерживая ее в этом положении, набирают код на малой клавиатуре. После отпускания клавиши **Alt** на экране появится соответствующий символ. Например, при нажатии клавиши **Alt** с последующим использованием в указанном порядке клавиш 1, 7, 2 и освобождением **Alt** на экране появится символ 1/4.

Однако нельзя получить непосредственно с клавиатуры в режиме немедленной обработки символ с кодом 127 или символ, код которого заключен между 0 и 31. Такие символы выводятся на экран дисплея программным способом; как это делается, мы покажем в гл. 8, а в гл. 11 опишем процедуры ввода этих символов с клавиатуры под управлением программ.

Числовые данные

В Бэйсике ПВМ числовые данные можно представлять пятью различными способами, каждый из которых имеет свое определенное назначение. Три типа представления дают возможность записывать числа с различной степенью точности; при этом увеличение точно-

Целое	*Восьмеричное*	*Шестнадцатеричное*
32767	&77777	&H7FFF
100	&144	&H64
23	&27	&H17
2	&2	&H2
0	&0	&H0
-1	-&1	-&H1
-187	-&273	-&HBB
-32767	&77777	-&H7FFF

С обычной точностью	*С двойной точностью*
1234567	1234567890123456
9999.99	99999999999999.99
123.123	12345678.123456789
.1234567	.12345678904123456
0	0
-.1234567	-.12345678904123456
-123.123	-12345678.123456789
-9999.99	-99999999999999.99
-1234567	1234567890123456

Рис. 7.1. Примеры различных типов числовых данных.

сти достигается за счет дополнительных затрат памяти и понижения скорости выполнения арифметических операций. Поэтому иногда выгоднее использовать типы данных, имеющие меньшую точность. Два других типа введены для удобства программирования более сложных задач.

Целый тип представляет собой запись без десятичной запятой числа в диапазоне от —32768 до 32767. Каждое значение целого типа занимает две ячейки памяти. При выполнении арифметических операций наибольшая скорость достигается, если операции производятся только над целыми числами. В Бэйсик-программе любое число без десятичной запятой рассматривается как целое при условии, что значение этого числа не выходит за указанные пределы.

Числа с плавающей запятой могут содержать десятичную запятую и дробную часть и записываться в обычном виде или в экспо-

Рис. 7.2. Экспоненциальное представление числовых данных.

ненциальной форме (рис. 7.2). Наибольшее допустимое число $1701412 \cdot 10^{32}$, а наименьшее отличное от нуля $0{,}5877471 \cdot 10^{-38}$. Существуют два типа представления чисел с плавающей запятой, которые отличаются друг от друга только степенью точности: в одном случае числа записываются с точностью до 6 знаков, а в другом — до 16.

Число с обычной точностью — это число с плавающей запятой, содержащее не более семи десятичных цифр. Для хранения числа в таком представлении используются четыре ячейки памяти. Следует иметь в виду, что арифметические действия выполняются операторами Бэйсика таким образом, что седьмую цифру нельзя считать достоверной, поэтому точными являются только первые 6 цифр. Например, число с обычной точностью, с четырьмя значащими цифрами до запятой, будет точным до сотых долей, а число с одной значащей цифрой до запятой представляется с точностью до пятого знака после запятой. В Бэйсик-программах всякое число, в конце которого стоит восклицательный знак, воспринимается как число с

обычной точностью; при этом автоматически производится округление до семи цифр.

Пример

```
Ok
print 1234.56789!
1234.568

—
```

Число с двойной точностью — это представление вещественного числа с точностью до 16 значащих цифр. Для хранения числа с двойной точностью используется 8 ячеек памяти. В действительности во внутримашинном представлении число с двойной точностью имеет 17 значащих цифр, но, поскольку последняя цифра не точна, на внешнем уровне (в программах) такое число всегда представляется 16 цифрами, точность которых гарантирована. Двойная точность требуется, например, для записи денежных сумм, выражаемых в долларах и центах, если соответствующие числа превышают $9999.99.

В Бэйсик-программах любое число, содержащее больше семи значащих цифр, автоматически рассматривается как значение с двойной точностью, если только в конце числа не стоит восклицательный знак. Для того чтобы некоторое число представлялось в машине с двойной точностью, надо поставить символ # за последней цифрой этого числа. Если в числе менее семи цифр, то при переводе его во внутренний код машины после последней цифры автоматически добавляются нули.

Шестнадцатеричные и восьмеричные числа

Обычно при программировании для записи чисел удобнее всего использовать привычную десятичную систему счисления с десятичными цифрами от 0 до 9. Внутри машины все числа представляются в системе счисления с основанием 2. Такие числа записываются в виде последовательностей нулей и единиц и называются *двоичными*. Бэйсик-интерпретатор автоматически выполняет преобразование чисел из десятичной системы в двоичную, так что программист может всегда пользоваться привычной ему десятичной системой. Это преобразование требует выполнения целого ряда арифметических действий (нельзя, лишь взглянув на десятичное число, сразу указать его двоичный эквивалент).

Некоторые программисты (особенно те, кто знаком с ассемблером) считают, что для определенного рода задач удобнее использовать систему счисления, которая легко преобразуется в двоичную. В качестве таких систем счисления чаще всего используют систему с основанием 8 *(восьмеричная)* и систему с основанием 16 *(шестнадцатеричная)*. Например, код ASCII иногда задается в шестнадцате-

ричном представлении. При преобразовании в двоичное число каждая цифра восьмеричного (или шестнадцатеричного) числа непосредственно заменяется соответствующим двоичным числом.

Таблица 7.1. **Представление чисел в шестнадцатеричной, восьмеричной и двоичной системах**

Шестнадцатеричное представление	Восьмеричное представление	Двоичное представление	Шестнадцатеричное представление	Восьмеричное представление	Двоичное представление
0	0	0000	8	10	1000
1	1	0001	9	11	1001
2	2	0010	A	12	1010
3	3	0011	B	13	1011
4	4	0100	C	14	1100
5	5	0101	D	15	1101
6	6	0110	E	16	1110
7	7	0111	F	17	1111

В табл. 7.1 показано, какое двоичное число соответствует каждой шестнадцатеричной и каждой восьмеричной цифре, а на рис. 7.3 поясняется, как можно легко преобразовать произвольное восьме-

10101001 Двоичное число	10101001 Двоичное число
A 9 Шестнадцатеричное число	2 5 1 Восьмеричное число
A. Для преобразования двоичного числа в шестнадцатеричное следует разбить все цифры двоичного представления на группы по 4 цифры, а затем воспользоваться табл. 7.1	B. Для преобразования двоичного числа в восьмеричное следует разбить все цифры двоичного представления на группы по 3 цифры (мысленно добавив, если надо, нули впереди числа), а затем воспользоваться табл. 7.1

Рис. 7.3. Преобразование шестнадцатеричных чисел в восьмеричные и двоичные.

ричное или шестнадцатеричное число в соответствующее двоичное представление или произвести обратное преобразование. Для преобразования восьмеричных или шестнадцатеричных чисел в десятичные и обратно, как и в двоичном случае, требуется выполнение достаточно сложных вычислений; для этого можно использовать переводную таблицу, специальный карманный калькулятор или программу.

Чтобы число в Бэйсик-программе воспринималось как восьмеричное целое, надо перед первой цифрой этого числа набрать символ &.

Пример

```
print &40
32
Ok
```

Для записи чисел в шестнадцатеричной системе наряду с десятичными цифрами от 0 до 9 используют буквы A, B, C, D, E и F. Для задания шестнадцатеричного числа перед первой его цифрой вводятся амперсанд (&) и буква H.

Пример

```
print &hda0
3488
Ok
```

Бэйсик не располагает средствами представления дробных восьмеричных или шестнадцатеричных чисел; допустимы лишь целые числа, изменяющиеся в пределах от —&77777 до &77777 (для восьмеричных чисел) и от —&H7FFF до &H7FFF (для шестнадцатеричных чисел). Более подробное описание двоичной, восьмеричной и шестнадцатеричной систем счисления можно найти в книге "Microcomputer Primer" (Mitchell Waite and Michael Pardee, Howard W. Sams & Co., Inc.) [1].

Переменные

Возможности вычислительной машины были бы слишком ограничены, если бы при работе с ней можно было использовать только постоянные величины (константы). Поэтому в любом языке програм-

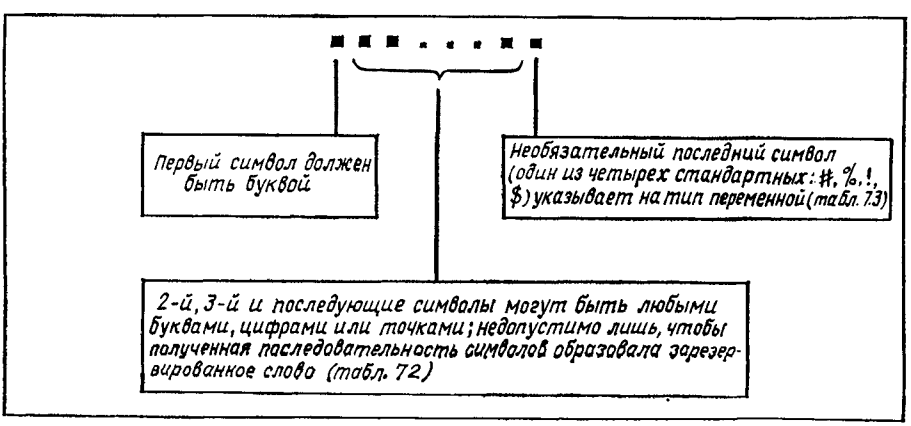


Рис. 7.4. Правила образования имен переменных в Бэйсике ПВМ.

[1] Такие сведения содержатся в любой книге по основам вычислительной техники.— *Прим. ред.*

мирования есть средства, позволяющие выделить часть памяти машины для хранения изменяющихся значений. Каждое такое значение идентифицируется уникальным именем, которое в совокупности с определенным значением называется *переменной*. Имя переменной в Бэйсике выбирается по определенным правилам (рис. 7.4) и может быть любой длины при условии, что оно умещается в одной строке программы. Следует, однако, помнить, что интерпретатор учитывает только первые 40 символов имени и игнорирует все остальные. В табл. 7.2 перечислены слова, которые нельзя использо-

Таблица 7.2. **Зарезервированные слова (запрещенные к использованию в качестве имен переменных)** [1]

ABS	CVS	FRE	LOG	POINT	SQR
AND	DATA	GET	LPOS	POKE	STEP
ASC	DATE$	GOSUB	LPRINT	POS	STICK
ATN	DEF	GOTO	LSET	PRESET	STOP
AUTO	DEFDBL	HEX$	MERGE	PRINT	STR$
BEEP	DEFINT	IF	MID$	PRINT #	STRIG
BLOAD	DEFSNG	IMP	MKD$	PSET	STRING$
BSAVE	DEFSTR	INKEY$	MKI$	PUT	SWAP
CALL	DELETE	INP	MKS$	RANDOMIZE	SYSTEM
CDBL	DIM	INPUT	MOD	READ	TAB(
CHAIN	DRAW	INPUT #	MOTOR	REM	TAN
CHR$	EDIT	INPUT$	NAME	RENUM	THEN
CINT	ELSE	INSTR	NEW	RESET	TIMES
CIRCLE	END	INT	NEXT	RESTORE	TO
CLEAR	EOF	KEY	NOT	RESUME	TROFF
CLOSE	EQV	KILL	OCTS	RETURN	TRON
CLS	ERASE	LEFT$	OFF	RIGHT$	USING
COLOR	ERL	LEN	ON	RND	USR
COM	ERR	LET	OPEN	RSET	VAL
COMMON	ERROR	LINE	OPTION	RUN	VARPTR
CONT	EXP	LIST	OR	SAVE	WAIT
COS	FIELD	LLIST	OUT	SCREEN	WEND
CSNG	FILES	LOAD	PAINT	SGN	WHILE
CSRLIN	FIX	LOC	PEEK	SIN	WIDTH
CVD	FN [2]	LOCATE	PEN	SOUND	WRITE
CVI	FOR	LOF	PLAY	SPACE$	WRITE #
				SPC(	XOR

[1]) Зарезервированное слово может быть частью длинного имени переменной; однако если к одиночному зарезервированному слову приписать в конце один из стандартных определяющих тип символов, то полученное в результате слово все равно не может служить именем переменной.

[2]) Сочетание букв FN разрешается использовать где угодно, но только не в качестве начальных букв имени, поскольку любое имя, начинающееся с FN, воспринимается интерпретатором как имя функции, определяемой пользователем.

вать в качестве имен переменных, поскольку они зарезервированы для командных и других ключевых слов. Вместе с тем любое зарезервированное слово может быть использовано как часть имени переменной. Например, имя DATAVAL допустимо, хотя и DATA, и VAL — зарезервированные слова и по отдельности именами переменных служить не могут.

Типы переменных

Каждому типу данных — строковому, целому, с обычной и с двойной точностью — соответствует определенный тип переменной. Тип переменной определяется последним символом ее имени. Соответствие между типами переменных и определяющими их символами указано в табл. 7.3. Имена, отличающиеся друг от друга лишь пос-

Таблица 7.3. **Типы переменных**

Тип	Символ, определяющий тип (стоит последним в имени) [1]	Количество байт	Диапазон значений
Целый	%	2	от —32768 до 32767
С двойной точностью	#	8	от —1.701411733192644D+38 до 1.701411733192644D+38
С обычной точностью	!	4	от —1.701412E+38 до 1.701412E+38
Строковый	$	2+ [2]	от 0 до 255 символов

[1] Если последний символ имени переменной не является одним из перечисленных стандартных символов, определяющих тип, то такая переменная будет считаться вещественной с обычной точностью; это соглашение можно изменить с помощью оператора DEF (гл. 8).

[2] Длина строковой переменной равна 2 байт плюс количество символов значения переменной.

ледним определяющим тип символом, воспринимаются интерпретатором как имена различных, совершенно несвязанных переменных. Так, A$, A!, A# и A% — это имена четырех разных переменных.

Всякий раз, когда интерпретатор встречает в программе новое имя переменной, он выделяет определенную область памяти для хранения ее значений соответственно определяемому типу (табл. 7.3).

Обычно если имя переменной не оканчивается одним из специальных символов, определяющих ее тип (%, #, $, !), то по принципу умолчания считается, что имя принадлежит числовой переменной с обычной точностью. Это соглашение нарушается в случае использования четырех модификаций оператора DEF, пользуясь которыми можно задавать тип переменной одной начальной буквой ее имени без применения классифицирующих суффиксов.

Пример. Определение целых переменных с помощью модификаций оператора DEF.

```
10 DEFINT A,J—M
```

В программе, содержащей указанный оператор, любая переменная, имя которой начинается с A, J, K, L или M, является переменной целого типа (независимо от того, оканчивается ли ее имя знаком %). Так, имена AARD, J1, K, LOOP, MARK будут определяться как целые переменные. Аналогично с помощью операторов DEFSTR, DEFSNG, DEFDBL можно задавать начальные буквы имен строковых переменных и переменных с обычной и двойной точностью соответственно. Однако действие оператора DEF в любой его модификации отменяется, если тип переменной указан в явном виде, т. е. имя содержит один из специальных суффиксов, определяющих тип переменной (%, #, $, !).

Выполнение оператора DEF должно предшествовать первому использованию определяемых им переменных. Этот оператор присваивает начальные значения всем переменным, имена которых начинаются с указанных в нем букв: числовым переменным присваивается значение нуль, а строковым — нулевая длина строки. Однако действие оператора DEF не распространяется на переменные, имена которых содержат стандартные, определяющие тип суффиксы.

Присваивание значений

Всякая переменная в программе должна принимать некоторые значения, поэтому интерпретатор автоматически присваивает каждой новой переменной (т. е. встретившейся в первый раз) значение, равное нулю (для числовых переменных) или нулевой строке (для строковых переменных). Кроме того, в Бэйсике ПВМ предусмотрено несколько операторов, позволяющих присваивать переменным и другие значения: чаще всего это делается с помощью оператора LET. В развернутой записи этот оператор имеет следующий вид:

```
500 LET TAX.RATE=6.5
510 LET TAX.AUTH$= "Отдел кредитов"
```

Командное слово LET необязательно, и обычно программисты его опускают, упрощая вид оператора. Например,

```
500 TAX.RATE=6.5
510 TAX.AUTH$ = "Отдел кредитов"
```

Оператор LET присваивает переменной с именем, указанным слева от знака равенства, значение, записанное справа от этого знака. Переменная и присваиваемое ей значение должны быть совместимы по типу данных. Если это условие не выполняется и, на-

пример, переменной строкового типа присваивается числовое значение, то будет выдано сообщение об ошибке "Type mismatch" («Несоответствие типов»).

Преобразование числовых типов данных

Числовое значение двойной точности, например 1.2345678, присваиваемое переменной, которая имеет обычную точность или является целой (например, A! или B%), округляется в соответствии с типом переменной. Аналогично, если значение обычной точности присваивается целой переменной (скажем, значение 1.2 присваивается переменной N%), это значение округляется до ближайшего целого числа.

Пример

```
a! = 1034500.89
Ok
print a!
1034501
Ok
```

В тех случаях, когда целое значение, например 253, присваивается переменной с обычной или двойной точностью (например, CD# или CD!), это значение дополняется десятичной запятой и следующими за ней нулями. Аналогично преобразуются значения обычной точности в случае присваивания их переменным с двойной точностью, но при этом преобразованное значение содержит не более шести точных цифр.

Массивы

В Бэйсике предусмотрены средства, позволяющие некоторым стандартным образом давать имена сразу целому массиву переменных. Вместо того чтобы присваивать уникальное имя каждой отдельной переменной, можно задать одно имя для целого их массива, а каждый элемент этого массива идентифицировать с помощью числового индекса.

Пример. Допустим, что надо написать программу для решения задачи, связанной с анализом цен на различные виды изделий. Каждому виду изделий можно поставить в соответствие отдельное уникальное имя переменной, значением которой является стоимость изделий данного вида (рис. 7.5). Однако в этом случае всякий раз, когда поступают сведения о новом виде изделий, пришлось бы вводить в программу новое имя переменной, а это, как правило, приводит к переписыванию почти всей программы. Вместо этого можно хранить значения стоимостей всех видов изделий в одном массиве, а стоимость конкретного вида изделий определять с помо-

щью индекса массива (рис. 7.6). В дальнейшем будет показано, что в подобных ситуациях значительно удобнее использовать массивы.

Для обращения к отдельному элементу массива достаточно после имени массива написать индекс, заключенный в скобки. Индексом может служить не только константа, но и переменная, что дает

Рис. 7.5. Использование разных имен для хранения значений.

возможность задавать действия, относящиеся к любому элементу массива.

Пример

```
10 A=12.3
20 B(J)=12.3
```

При выполнении оператора строки 10 значение 12.3 присваивается всегда одной и той же переменной A. При выполнении второго

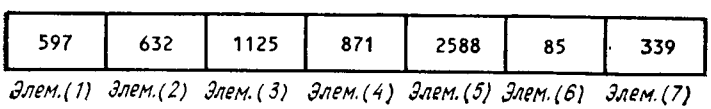


Рис. 7.6. Хранение значений в массиве.

оператора (строка 20) это значение может быть присвоено любому элементу массива B; какой именно элемент принимает значение 12.3, определяется текущим значением переменной J.

Правила образования имен массивов в Бэйсике ПВМ те же, что и для имен простых переменных (рис. 7.4 и табл. 7.2). В одной и той же программе допустимы одинаковые имена массивов и простых переменных; в этом случае они считаются никак не связанными друг с другом, как если бы они имели различные имена. Все элементы одного массива имеют один и тот же тип — строковый, целый, с обычной или с двойной точностью. Действие операторов DEF, в которых определяются начальные буквы имен переменных заданного типа, распространяется и на имена массивов. Точно так же, как и в случае простых переменных, тип массива может быть

задан явным образом — для этого в конце имени массива ставится один из стандартных символов, определяющих тип (%, #, $, !); соответствие между этими символами и типами данных такое же, как и в случае простых переменных. Как и для простых переменных, стандартный символ определения типа, стоящий в конце имени массива, отменяет действие оператора DEF, относящееся к этому имени.

Размерности массивов

В Бэйсике ПВМ массив может иметь несколько индексов. Массив с одним индексом можно представить себе как одномерный столбец (или одномерную строку) значений (рис. 7.6). Использова-

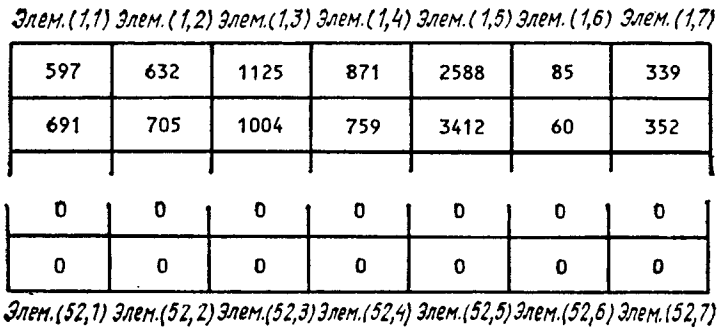


Рис. 7.7. Хранение значений в двумерном массиве.

ние второго индекса аналогично добавлению второго измерения (рис. 7.7).

Пример. Допустим, что необходимо фиксировать стоимость различных видов изделий каждую неделю в течение всего года. Для этого можно было бы использовать 52 отдельные переменные для каждого вида изделий, однако работать с таким огромным количеством различных имен переменных очень неудобно. Поэтому лучше использовать 52 одномерных массива, а еще лучше — один двумерный массив. Первый индекс такого двумерного массива мог бы указывать на конкретный вид изделий, а второй — на конкретную неделю.

Один массив может иметь три, четыре, пять и более индексов. Трехмерный массив можно представить себе как куб, а для бо́льших размерностей трудно найти простой геометрический образ. Максимально допустимое число индексов для одного массива — 255.

Задание размерности массива

Если в программе предполагается использовать одномерный массив, содержащий более 10 элементов, или какой-либо многомерный массив, то прежде всего необходимо его описать, т. е. определить количество индексов и максимальное значение каждого из них. Для задания размерности массивов предназначен оператор DIM, в котором перечисляются имена массивов и указываются максимальные значения их индексов.

Пример. Определение четырехмерного массива строкового типа с максимальными значениями индексов 4, 8, 2 и 50.

```
110 DIM S$(4,8,2,50)
```

В одном операторе DIM можно описывать несколько массивов. Для этого после командного слова DIM следует перечислить описания различных массивов, отделяя их друг от друга запятыми.

Пример

```
10 DIM ITEM$(35), COST#(2,35), UNIT%(2,35)
```

Если в программе имеется оператор DEF, действие которого распространяется на имена массивов, перечисленных в операторе DIM, то он должен обязательно предшествовать оператору DIM. В этом случае при выполнении оператора DIM для каждого указанного в нем массива будет отведен достаточный объем динамической памяти, всем элементам числовых массивов будут присвоены начальные значения, равные нулю, а всем элементам строковых массивов — нулевые строки. Если для размещения всех массивов объем памяти оказывается недостаточным, выдается сообщение "Out of memory" («Нехватка памяти»), и оператор DIM выполняется только для первых перечисленных в нем массивов, которые полностью уместились в памяти.

Оператор DIM можно также использовать и в режиме немедленной обработки. Однако в этом случае с определяемыми в этом операторе массивами нельзя работать в программируемом режиме после выполнения команды RUN, которая уничтожает все описания массивов.

Изменение размерностей массивов

Определив размерность массива посредством оператора DIM, ее уже нельзя изменить с помощью некоторого другого оператора DIM. Для этого необходимо сначала удалить соответствующий массив из памяти с использованием стирающего оператора ERASE. Этот последний состоит из командного слова ERASE и следующего за ним списка имен массивов. Имена массивов в этом списке перечисляются через запятую, причем максимальные значения индексов массивов не указываются.

Пример

```
10 DIM ITEM$(40),COST(40),PRICE(40)
20 REM
  .      .
  .      .
  .      .
90 REM
100 ERASE PRICE,COST
110 DIM  COST (2,40)
120 REM
  .
  .
  .
```

При выполнении нового оператора DIM, переопределяющего структуру некоторого массива, всем элементам этого массива присваиваются нулевые начальные значения.

Наименьшее значение индекса массива

По умолчанию, т. е. если не предпринимается специальных действий, всякий индекс массива изменяется, начиная с нуля. Однако часто удобнее и привычнее работать с массивом, изменяя его индекс от единицы. Для того чтобы первому элементу массива действительно соответствовало значение индекса, равное единице, и тем самым не тратилась зря память, отводимая под элемент с индексом, равным нулю, можно использовать следующий оператор:

```
100 OPTION BASE 1
```

С помощью оператора OPTION BASE устанавливается наименьшее значение индексов массивов 0 или 1. Этот оператор должен выполняться в программе до первого оператора, содержащего имена массивов, включая операторы DEF и DIM. В противном случае будет выдано сообщение "Duplicate Definition" («Повторное определение»). Оператор OPTION BASE можно использовать как в режиме немедленной обработки, так и в программируемом режиме, однако перед выполнением программы по команде RUN наименьшее значение индекса всегда устанавливается равным нулю. Таким образом, команда OPTION BASE 1, выполненная в режиме немедленной обработки, не окажет никакого влияния на массивы, используемые в программе. Это значит, что если индексы в программе должны меняться, начиная с единицы, то программа должна содержать свой собственный оператор OPTION BASE.

Глава 8

РАБОТА С ЧИСЛОВЫМИ И СТРОКОВЫМИ ДАННЫМИ

Существует множество способов манипулирования константами, переменными, массивами и выполнения над ними различных совместных операций. В данной главе описывается ряд дополнительных методов присваивания значений переменным, а также демонстрируется, как с помощью выражений и функций можно комбинировать различные данные, выполнять преобразования одних типов данных в другие и отображать требуемые значения на экране дисплея.

Присваивание значений переменным

Наиболее распространенным средством однократного присваивания значения переменной является оператор LET (гл. 7). Однако он оказывается неудобным в случае присваивания значений большому числу переменных и ввода присваиваемых значений с клавиатуры в процессе выполнения программы. Поэтому для выполнения этой операции в Бэйсике ПВМ предусмотрены специальные, более эффективные операторы.

Операторы DATA и READ

Эти операторы позволяют просто и эффективно осуществить операцию присваивания постоянных значений большому числу переменных и элементам массивов. При этом операторы DATA создают список постоянных значений, а операторы READ присваивают эти значения переменным и элементам массивов.

Пример

```
list
99 'Таблица номеров счетов
100 DATA 100, Ферн Герцберг, 150, Оги Богдис,
    200, Ксавьер Декото
1139 'Присвоить номер счета и имя
1140 READ acct(1),cust$(1),acct(2), cust$(2),
     acct(3), cust$(3)
Ok
```

Оператор DATA может содержать любые допустимые числовые (рис. 7.1) или строковые константы. Используемая строковая константа заключается в кавычки, если она начинается со значащих пробелов или кончается ими либо если один из входящих в нее символов — запятая или двоеточие. При этом заключенная в кавычки строка не должна содержать кавычек.

Если в программе встречается несколько операторов DATA, то все они участвуют в формировании одного и того же списка значений. Этот список создается последовательно, начиная со значений, указанных в первом операторе DATA (расположенном первым в тексте программы), и кончая значениями, относящимися к последнему из них (рис. 8.1). Операторы DATA не относятся к числу исполняемых.

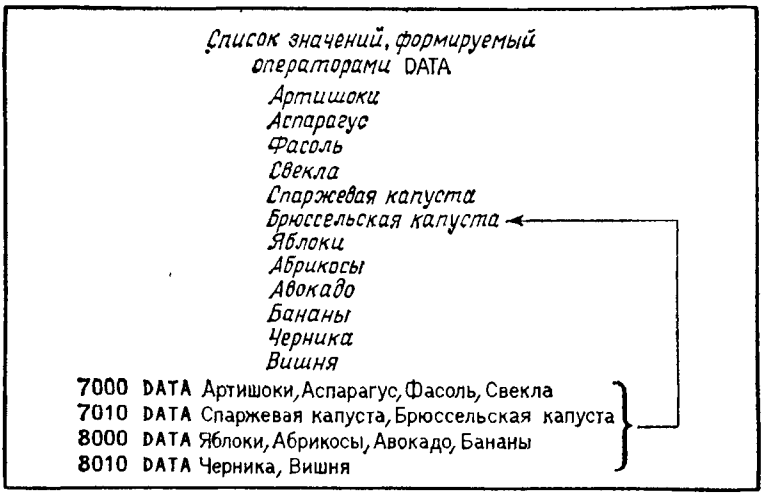


Рис. 8.1. Все операторы DATA формируют единый список значений.

Первый выполняемый оператор READ присваивает первое значение из списка, формируемого операторами DATA, первой переменной. Второй переменной оператора READ присваивается второе значение, третьей — третье и т. д. до тех пор, пока не будут присвоены значения всем переменным этого оператора. Если в дальнейшем в программе выполняется еще один оператор READ, он берет присваиваемые значения из того же списка, начиная с позиции, перед которой закончил работу предыдущий оператор READ (рис. 8.2).

Неиспользованные значения из списка DATA игнорируются, а если при выполнении некоторого оператора READ оказалось, что список уже просмотрен до конца, то выдается сообщение об ошибке "Out of DATA..." («Нет данных...»). Типы переменных и элементов массивов, указанных в операторе READ, должны быть совместимы с типами присваиваемых им значений. Так, недопустимо, чтобы переменная была строковой, а значение — числовым (или наоборот). В случае подобного несоответствия выдаются сообщение "Syntax error..." («Синтаксическая ошибка...») и текст оператора READ, в котором обнаружилось несоответствие типов данных. Если и переменная, и значение являются числовыми, но их типы

различаются по точности, оператор READ выполняет преобразования, аналогичные действию оператора LET (гл. 7).

При использовании оператора DATA в режиме немедленной обработки ошибок не возникает, однако перечисленные в нем значения окажутся недоступными как в режиме немедленной обработки, так и в программируемом режиме; объясняется это тем, что в ре-

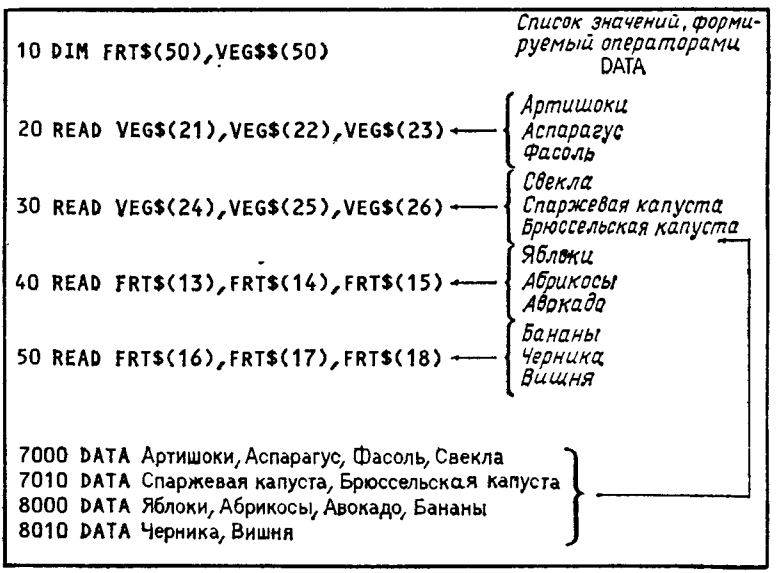


Рис. 8.2. Операторы READ присваивают переменным значения из списка, формируемого операторами DATA.

жиме немедленной обработки оператор READ выбирает очередное значение из списка, созданного к этому моменту операторами DATA программируемого режима.

Оператор RESTORE

Бэйсик-интерпретатор ПВМ отслеживает текущую позицию в списке значений DATA с помощью специального указателя, который перемещается вперед всякий раз, когда некоторый оператор READ считывает из списка очередное значение. Прямое управление перемещением этого указателя может быть осуществлено с помощью оператора RESTORE. В простейшем случае оператор RESTORE возвращает указатель к началу списка, формируемого операторами DATA; при этом очередной оператор READ, следующий после оператора RESTORE, будет использовать значения из списка повторно.

Оператор RESTORE позволяет также устанавливать указатель на элемент списка, соответствующий любому промежуточному

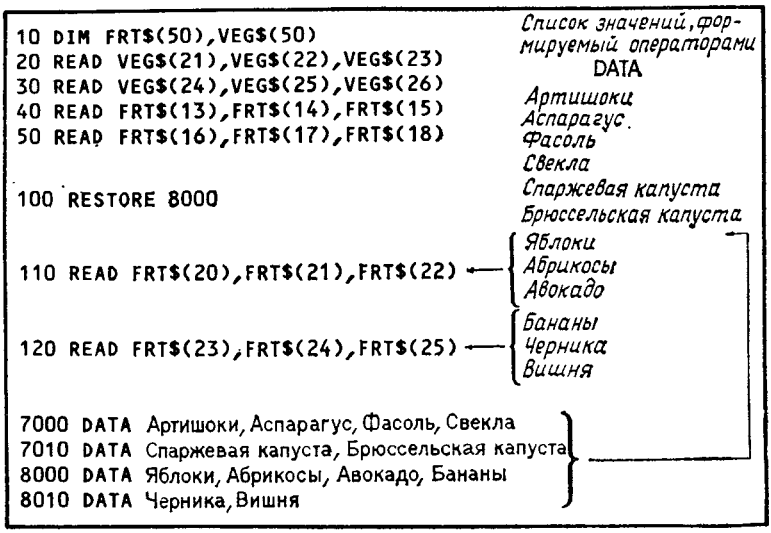


Рис. 8.3. Оператор RESTORE перемещает указатель в списке значений, формируемом операторами DATA (в данном случае он настроен на строку 8000).

оператору DATA. Для этого в операторе RESTORE надо указать номер строки требуемого оператора DATA. Если в строке с указанным номером обнаружится такой оператор, то будет выбран ближайший оператор DATA, расположенный в программе после указанной строки.

Ввод значений с клавиатуры

При программировании довольно часто значения строковых или числовых переменных должны быть заданы пользователями программ. Для этого в Бэйсике предусмотрен оператор INPUT, который считывает вводимые значения с клавиатуры и присваивает их переменным или элементам массивов. В простейшем случае этот оператор выглядит так:

```
350 INPUT A
```

Когда очередным выполняемым оператором оказывается оператор подобного вида, на экран дисплея выводится знак вопроса. Появление этого знака означает, что пользователь программы должен ввести с клавиатуры необходимую информацию. При этом клавиша ← обеспечивает перемещение курсора влево на один символ, а клавиша **Esc** стирает всю строку, включая знак вопроса, что позволяет начать ввод записи снова. Обе эти клавиши действуют до

тех пор, пока не нажимается клавиша ↵, указывающая на окончание ввода.

При ответе на запрос оператора INPUT строковое значение необходимо заключить в кавычки при условии, что оно содержит запятые либо начинается (или кончается) значащими пробелами. При этом заключенная в кавычки строка не должна содержать кавычек.

Наводящие сообщения оператора INPUT

При работе далеко не всех программ удобно, чтобы требование на ввод с клавиатуры тех или иных значений выводилось на экран в виде одного лишь знака вопроса. Оператор INPUT позволяет кроме знака вопроса (или вместо него) выводить некоторое наводящее сообщение. Текст этого сообщения должен указываться в операторе INPUT в виде строковой константы, стоящей непосредственно за командным словом, при этом строковая константа заключается в кавычки и отделяется от имени переменной точкой с запятой.

Пример. Наводящее сообщение в режиме немедленной обработки:

```
input "Срок займа в годах"; срок
Срок займа в годах? 10
Ok
```

Для того чтобы после наводящего сообщения не появился знак вопроса, точку с запятой в операторе INPUT следует заменить запятой. Тогда вводимая пользователем запись будет располагаться непосредственно за последним символом сообщения.

Пример

```
input "Срок займа в годах = ", срок
Срок займа в годах = 10
Ok
```

Ввод с клавиатуры нескольких значений

С помощью одного оператора INPUT можно ввести значения сразу нескольких переменных или элементов массивов. Для этого достаточно перечислить все эти переменные и элементы массивов в конце оператора INPUT, отделив их друг от друга запятыми. Как и в предыдущих случаях, при работе программы пользователю будет выведен либо только знак вопроса, либо наводящее сообщение, если оно было включено в оператор INPUT. В качестве ответа пользователь должен указать значения каждой переменной оператора INPUT, отделив их друг от друга запятыми.

Пример

```
list
10 INPUT "Имя и № в системе соцстраха";N$,SS#
Ok
run
Имя и № в системе соцстраха? Элвин Фокс, 123-45-678
Ok
```

Проверка правильности записей, вводимых с клавиатуры

После того как пользователь программы нажимает клавишу ↵, отмечая окончание ввода, производится проверка правильности вводимого сообщения. При необходимости выполняются преобразования числовых значений для согласования их точности с точностью соответствующих им переменных, подобно тому как это делается операторами LET и READ. Если количество набранных пользователем значений больше или меньше числа переменных и элементов массивов оператора INPUT либо если пользователь вводит строковое значение в качестве значения числовой переменной, то выдается сообщение "?Redo from start" («Повторите ввод»). После этого сообщения осуществляется повторный ввод всех значений с самого начала строки.

Взаимный обмен значениями между переменными

Оператор SWAP инициирует обмен значениями между двумя переменными.

Пример

```
COST1 = 5
Ok
COST2 = 21
Ok
SWAP COST1,COST2
Ok
PRINT COST1
21
Ok
PRINT COST2
5
Ok
```

Указанные в операторе SWAP переменные должны иметь один и тот же тип данных, в противном случае выдается сообщение «Type mismatch» («Несоответствие типов»). Переменные оператора SWAP (каждая в отдельности и обе вместе) могут быть как простыми переменными, так и элементами некоторого массива.

Обнуление значений переменных и элементов массивов

Оператор CLEAR обнуляет значения сразу всех переменных и элементов массивов, не затрагивая строк программы: числовые переменные становятся при этом равными нулю, а строковым придаются нулевые строковые значения. Кроме того, оператор CLEAR уничтожает результаты работы выполненных ранее операторов DIM и DEF.

Вывод данных

Оператор PRINT предназначен для вывода значений переменных и элементов массивов как в режиме немедленной обработки, так и в программируемом режиме.

Пример. Действие оператора PRINT при выводе числовых и строковых значений.

```
list
100 INPUT "Наименование изделия" ; ITEM$
110 INPUT "Стоимость" ; VALUE
200 PRINT
210 PRINT ITEM$; "стоимость в долл.";VALUE
Ok
run
Наименование изделия? Медный светильник
Стоимость? 83.50

Медный светильник стоимость в долл. 83.5
Ok
```

Этот пример демонстрирует некоторые особенности оператора PRINT. Так, в своем простейшем виде оператор PRINT (строка 200) обеспечивает вывод на экран дисплея пустой строки. Строка 210 показывает, что с помощью одного оператора PRINT можно вывести несколько значений (в примере выводятся строковая переменная, строковая константа и числовая переменная).

Вывод пробелов

Число пробелов между соседними словами определяется типами соответствующих значений, а также знаками препинания, используемыми в операторе PRINT при перечислении значений или имен переменных. Любые два значения, разделенные в операторе PRINT точкой с запятой, выводятся непосредственно друг за другом. При выводе таким образом строковых значений можно получить конкатенацию нескольких строк, и в ряде случаев это удобно. Однако слитное написание числовых значений практически всегда бессмысленно, и поэтому после каждого числового значения обяза-

тельно дается один пробел. Использование точки с запятой в операторе PRINT при выводе числовых и строковых значений было проиллюстрировано в предыдущем примере.

Замена в операторе PRINT точки с запятой на запятую позволяет выводить значения в стандартной табличной форме по столбцам. В этом случае оператор PRINT делит весь экран дисплея на несколько зон; ширина каждой зоны (кроме последней) равна 14 символам. Ширина последней зоны составляет 12 символов при 40-символьном экране и 10 символов при 80-символьном экране. Если в операторе PRINT перед именем переменной (или собственно значением) стоит запятая, то перед выводом соответствующего значения курсор переместится к началу очередной зоны.

Пример

```
list
50  PRINT "Наименование изделия", "— Стоимость —"
100   READ ITEM$,COST
110   PRINT ITEM$,,COST
120   READ ITEM$,COST
130   PRINT ITEM$,,COST
140   PRINT
1000   DATA Письменный стол,875,Стул,260
Ok
run

Наименование изделия          — Стоимость —
Письменный стол                      875
Стул                                 260
Ok
```

В данном примере первая выводимая строковая константа программы (строка 50) выходит за пределы первой зоны, и поэтому вторая константа выводится в третьей зоне. Для того чтобы последующие выводимые значения располагались в нужных столбцах, в соответствующих операторах PRINT (строки 110 и 130 программы) между именами переменных ставятся две запятые и при выводе вторая зона пропускается.

Установка исходной позиции курсора при выводе данных

При отсутствии в конце оператора PRINT запятой или точки с запятой выполнение этого оператора завершается перемещением курсора в крайнюю левую позицию экрана и на одну строку вниз. По аналогии с механизмом пишущих машинок и телетайпов это действие названо *возвратом каретки*.

Возврат каретки можно отменить, если поставить в конце оператора PRINT запятую или точку с запятой. Если оператор PRINT оканчивается точкой с запятой, то курсор остается в конце послед-

него выведенного значения, а если запятой, то курсор устанавливается в начало следующей зоны.

Пример. Используя точку с запятой в операторе PRINT, можно включить в наводящее сообщение значение некоторой переменной.

```
list
100 INPUT "Как Вас зовут"; N$;
110 PRINT "Сколько Вам лет, " ; N$;
120 INPUT AGE
Ok
run
Как Вас зовут? Иола
Сколько Вам лет, Иола? 32
Ok
```

В приведенной программе строковое значение (строка 100) сначала вводится пользователем, а затем это значение (строка 110) включается с помощью оператора PRINT во второе выводимое сообщение. Поскольку в конце оператора PRINT стоит точка с запятой, возврата каретки не происходит и ответ пользователя на запрос второго оператора INPUT (строка 120) располагается в той же строке экрана дисплея, что и само сообщение.

При выводе достаточно длинных строк возврат каретки производится автоматически при достижении курсором крайней правой позиции на экране дисплея. Автоматический возврат каретки происходит всегда независимо от знака пунктуации, стоящего в конце оператора PRINT. Это приводит иногда к неожиданному побочному эффекту: если в результате выполнения оператора PRINT заполняется в точности одна строка экрана, возврат каретки производится подряд два раза (если только в конце PRINT не стоит точка с запятой). В результате двойного возврата каретки на экране пропускается пустая строка, а если перед этим была заполнена последняя строка экрана, то весь текст на экране сдвигается на одну строку вверх.

Пример. Вывод значений на 40-символьный экран дисплея.

```
list
10 PRINT ,,"123456789012";
20 PRINT ,,"абвгдежзиклм"
30 PRINT ,,"123456789012";
Ok
run
                                123456789012
                                абвгдежзиклм
                                123456789012

Ok
```

В результате выполнения любого из трех указанных выше операторов PRINT одна строка экрана дисплея заполняется целиком. Поскольку первый из этих операторов оканчивается точкой с запятой, возврата каретки, сопровождающего обычно завершение оператора PRINT, не происходит, и осуществляется лишь автоматический возврат каретки после заполнения последней позиции в строке экрана. Поскольку в конце второго оператора PRINT не стоит точка с запятой, возврат каретки производится два раза подряд, в результате чего пропускается незаполненная строка.

Одна из особенностей оператора PRINT, связанная с выводом длинных строк, состоит в следующем. Если оказывается, что выводимое значение не умещается в оставшейся части текущей строки, то оператор PRINT выводит это значение целиком на следующей строке, начиная с крайней левой позиции.

Пример

```
print "A" ,, "1234567890123"
A
1234567890123
Ok
```

Две запятые в операторе PRINT приводят к перемещению курсора на 30-ю символьную позицию строки экрана, после чего оператор должен вывести 13-символьное строковое значение. Поскольку целиком это значение не умещается в оставшихся 10 позициях на текущей строке (рассматривается 40-символьный экран), оно выводится с начала следующей строки.

Выводимые числовые значения

Вид выводимых числовых значений зависит от целого ряда факторов. Если выводимое значение отрицательное, то перед самим значением выводится знак минус, а если положительное, то вместо знака плюс выводится (в качестве первого символа) пробел. Как при отрицательных, так и при положительных значениях после последней выведенной цифры оператор PRINT добавляет один пробел.

Пример

```
list
240  INPUT "Введите число: ",N
250 PRINT
260 PRINT N; "плюс "; —N; "равно 0"
Ok
run
Введите  число: 3
3 плюс —3  равно 0
Ok
```

Если в числе не слишком много значащих цифр, оно выводится в обычном десятичном представлении. Значения с обычной точностью, содержащие более семи значащих цифр, и значения с двойной точностью, имеющие более 16 значащих цифр, выводятся в экспоненциальном представлении (рис. 7.2).

Пример

```
list
1300 DEFSNG S:DEFDBL D
1400 INPUT "Значение с обычной точностью";SINGLE
1410 INPUT "Значение с двойной точностью";DOUBLE
1420 PRINT
1430 PRINT "Значение с обычной точностью: ";SINGLE
1440 PRINT "Значение с двойной точностью: ";DOUBLE
Ok
run
Значение с обычной точностью? 12345678900
Значение с двойной точностью? .0012345678901234567 89

Значение с обычной точностью: 1.234568E+10
Значение с двойной точностью: 1.2345678901234 57E—03
Ok
run
Значение с обычной точностью? 1234.56789
Значение с двойной точностью? 123456789.123456789

Значение с обычной точностью: 1234.568
Значение с двойной точностью: 123456789.1234568
Ok
```

Стоящий после числовой константы восклицательный знак указывает на обычную точность, знак процента (%) — на целый тип значения, в остальных случаях числовая константа считается значением с двойной точностью. Все числовые значения выводятся в десятичной форме (восьмеричные и шестнадцатеричные константы перед выводом преобразуются).

При выполнении оператора PRINT не выводятся нули, стоящие в начале числа до десятичной запятой, и нули, стоящие после запятой в конце всего числа, что не очень удобно при выводе значений денежных сумм (например, стоимость 83.50 будет иметь вид 83.5). Для вывода значений денежных сумм в удобном виде в Бэйсике предусмотрен оператор PRINT USING (гл. 10).

Выражения над переменными и массивами

При составлении программ часто возникает необходимость комбинировать и сравнивать значения различных переменных, элементов массивов и констант для самого разнообразного их последую-

щего использования. Такие сочетания и сравнения осуществляются с помощью *выражений.*

Выражение состоит из *операндов* (тех значений, которые сравниваются или комбинируются) и *знаков операций*, определяющих способ комбинирования или сравнения. Наряду с обычными арифметическими операциями сложения, вычитания, умножения и деления в выражения могут входить и более сложные операции над числами. Кроме того, используются *операции сравнения* (значений двух операндов), *логические операции*, которые соединяют операнды соответственно правилам логики, а также одна *строковая операция.*

Выражения можно использовать практически во всех тех случаях, когда переменные или константы служат для вычисления некоторых значений. Например, с помощью выражения можно задать выводимое значение в операторе PRINT, определить значение индекса для элемента массива, установить размерность массива в операторе DIM. Ограничения, накладываемые на использование выражений, как правило, довольно естественны. Так, выражениям нельзя присваивать значения с помощью операторов LET, READ или INPUT. Выражения, перечисленные в качестве значений в операторе DATA, воспринимаются интерпретатором как строковые константы, т. е. не вычисляются. И наконец, тип значения, получаемого в результате вычисления всего выражения, должен быть согласован с характером использования этого выражения. Например, выражение, определяющее строковое значение, нельзя использовать в качестве индекса массива.

Приоритет операций

В программах часто присутствуют сложные выражения, содержащие большое число операций и операндов. При вычислении таких выражений интерпретатору должно быть известно, в какой последовательности нужно выполнять входящие в них операции. Однако обычно результат вычислений зависит от порядка выполнения операций. Например, если при вычислении выражения 100+ +10*5 сначала выполняется умножение, то результат равен 150 (10*5=50 и 100+50=150), а если сначала выполняется сложение, то в результате получится 550 (100+10=110 и 110*5=550). Однако в некоторых случаях порядок выполнения операций выражения несуществен. Например, при вычислении выражения 6—3+1—23 неважно, какие операции выполнять сначала, сложение или вычитание; результат будет одним и тем же: —19.

В Бэйсике ПВМ выражения вычисляются в соответствии со стандартными правилами операторного предшествования. Приоритеты различных операций приведены в табл. 8.1. Согласно этой таблице, умножение имеет больший приоритет, чем сложение, так что результатом выражения 100+10*5 будет 150; некоторые операции

Таблица 8.1. **Стандартные приоритеты операций**

Обозначение операции	Приоритет [1]	Число операндов	Название операции
		Числовые операции [2]	
∧	12	2 числовых	Возведение в степень
—	11	1 числовой	Изменение знака
*	10	2 числовых	Умножение
/	10	2 числовых	Деление
\	9	2 числовых	Деление нацело [3]
MOD	8	2 числовых	Вычитание по модулю [4]
+	7	2 числовых	Сложение
—	7	2 числовых	Вычитание
		Строковые операции	
+	7	2 строковых	Конкатенация
		Операции сравнения [5]	
=	6	2 однотипных	Равно
<>	6	2 однотипных	Не равно
><	6	2 однотипных	Не равно
<	6	2 однотипных	Меньше чем
>	6	2 однотипных	Больше чем
<=	6	2 однотипных	Меньше или равно
=<	6	2 однотипных	Меньше или равно
>=	6	2 однотипных	Больше или равно
=>	6	2 однотипных	Больше или равно
		Логические операции [6]	
NOT	5	1 логический	Отрицание
AND	4	2 логических	Конъюнкция
OR	3	2 логических	Дизъюнкция
XOR	2	2 логических	Исключающая дизъюнкция
EQV	2	2 логических	Эквивалентность
IMP	1	2 логических	Импликация

[1]) Операции перечислены в порядке убывания приоритета; операции с одинаковым приоритетом выполняются последовательно слева направо. Стандартный приоритет операций можно изменить с помощью скобок: выражения в скобках всегда вычисляются в первую очередь.

[2]) Если числовые операнды одного и того же выражения различаются по точности, то среди них выбирается операнд с наибольшей точностью, а все остальные операнды преобразуются к его типу.

[3]) Значения операндов при делении нацело должны быть в пределах от —32768 до 32767; для получения результата этой операции отбрасывается остаток от деления.

[4]) Эта операция дает остаток от деления значения первого операнда на значение второго.

[5]) В операции сравнения операнды должны быть совместимыми по типу, т. е. либо оба числовые, либо оба строковые.

[6]) Подробно логические операции описаны в табл. 8.2. Операндами логических операций могут быть выражения с операциями сравнения или числовые значения от —32768 до 32767.

имеют одинаковый приоритет. Если в одном и том же выражении встречается несколько равноприоритетных операций, то первой выполняется крайняя левая операция, затем — вторая слева и т. д. Например, результат вычисления выражения 200/5—4*10 равен 0, поскольку 200/5=40, 4*10=40 и 40—40=0.

Изменение стандартного порядка операций

Стандартный порядок выполнения операций можно изменить с помощью скобок, так как первыми всегда выполняются операции, заключенные в скобки. Например, 200/(5—4)*10 равно 2000, так как (5—4)=1, 200/1=200 и 200*10=2000. В пределах скобок операции выполняются в стандартном порядке, если только он не нарушается наличием внутренних скобок. Скобки могут вкладываться одни в другие практически неограниченно; первым всегда выполняется подвыражение, заключенное в самые внутренние скобки. Хотя существует теоретический предел числа вложений скобок, который зависит от степени сложности выражения, на практике он достигается весьма редко. Если все же выражение не удовлетворяет этому ограничению, выдается сообщение "Out of memory..." («Нехватка памяти...»).

Числовые выражения

В числовые выражения входят операции над целыми, вещественными с обычной и двойной точностью, восьмеричными и шестнадцатеричными значениями. Если числовое выражение содержит операнды различных типов, то при его вычислении среди них выделяется тип, имеющий наибольшую точность, и все остальные операнды преобразуются к этому типу. С той же точностью вычисляется и результат всего выражения. Заметим, что это преобразование не повышает действительной точности операндов. Например, целое значение, преобразованное в вещественный тип данных, всегда дополняется нулевой дробной частью (все значащие цифры дробной части преобразованного числа равны нулю). Аналогично значение с обычной точностью при преобразовании к двойной точности будет иметь не более шести значащих цифр, отличных от нуля.

Окончательная точность результата числового выражения определяется характером его использования. Если выражение входит в оператор PRINT, то его значение выводится с той же точностью, с которой оно было вычислено; если же результат выражения присваивается некоторой переменной, то он преобразуется так, чтобы соответствовать точности этой переменной. При использовании выражения в качестве индекса массива результат его вычисления округляется до целого значения.

Числовые операции, допустимые в Бэйсике ПВМ, описаны в табл. 8.1. Всего таких операций восемь: четыре из них — это

обычные арифметические действия сложения, вычитания, умножения и деления, а остальные четыре — менее распространенные операции возведения в степень, перемены знака, деления нацело и вычисления остатка. При выполнении операции *возведения в степень* значение умножается само на себя заданное число раз. Операция *перемена знака* преобразует заданное число в число с той же абсолютной величиной, но противоположного знака. *Деление нацело* выполняется как обычное деление, только операнды при этом должны быть не меньше —32768 и не больше 32767, а дробная часть результата деления отбрасывается (не округляется), и остается лишь целая часть частного. Операция *вычисление остатка* позволяет получить остаток от деления одного числа на другое.

Пример

```
list
10 INPUT "1-е число";N1
20 INPUT "2-е число";N2
30 PRINT "Возведение в степень: ";
   N1; "^";N2; "=";N1^N2
40 PRINT "Перемена знака: —"; N1; "="; —N1
50 PRINT "Деление нацело: ";
   N1; "\";N2; "=";N1\N2
60 PRINT "Вычисление остатка: ";
   N1; "MOD";N2; "=";N1 MOD N2
70 RUN 'Рестарт; для останова нажать <Break>
Ok
run
1-е число? 10
2-е число? 3
Возведение в степень: 10 ^ 3 = 1000
Перемена знака: — 10 =—10
Деление нацело: 10\3 = 3
Вычисление остатка: 10 MOD 3 = 1
1-е число?
Break in 10
Ok
```

Заметим, что в приведенном примере повторный запуск программы производится с помощью команды RUN (строка 70). Для прерывания выполнения программы и перехода к режиму немедленной обработки следует использовать комбинацию операторов **Ctrl | Scroll | Lock** (клавиша Break).

Строковые выражения

В наборе операций Бэйсика ПВМ имеется одна чисто строковая операция — конкатенация, т. е. соединение строковых значений друг с другом с образованием одной длинной строки.

Пример. Выполнение операции конкатенации.

```
list
10 INPUT "1-е строковое значение";S1$
20 INPUT "2-е строковое значение";S2$
30 S$=S1$+S2$
40 PRINT S1$; " + ";S2$;" = ";S$
Ok
run
1-е строковое значение? пере
2-е строковое значение? оценка
пере + оценка = переоценка
Ok
```

Выражения с операциями сравнения

Операция сравнения позволяет сравнивать два значения и устанавливать, выполняется ли между ними заданное соотношение. Допустимы следующие шесть типов отношений: равно, не равно, меньше, меньше или равно, больше, больше или равно. Соответствующие операции сравнения приведены в табл. 8.1.

Если операция сравнения выполняется над числовыми значениями, то эти значения могут быть разного типа. Однако при сравнении значений, имеющих различную точность, следует иметь в виду некоторые особенности представления чисел в памяти вычислительной машины. Например, представление числа 1,23456789 в виде значения с обычной точностью меньше, чем представление этого же числа в виде значения с двойной точностью. На первый взгляд может показаться, что первое значение должно быть больше второго, поскольку значение с обычной точностью 1,234568 (значение числа, округленное до 6 знаков после запятой) больше, чем 1,23456789. Однако следует помнить, что седьмая значащая цифра значения с обычной точностью считается ненадежной и при сравнении игнорируется. Таким образом, в действительности сравниваются значения 1,23456789 и 1,23456000.

Для сравнения двух числовых значений лучше всего вычесть одно из другого и сравнить разность с нулем. Если при этом хотя бы одно из значений целое, то числа можно считать равными при условии, что разность отличается от нуля меньше чем на 1; если среди сравниваемых значений нет целых, но есть хотя бы одно с обычной точностью, то разность должна отличаться от нуля меньше, чем на .000001, а если оба значения с двойной точностью, то меньше чем на 1E-16. Например, вместо A#=B! лучше писать ABS (A#—B!)<.000001.

Строковое значение можно сравнивать только с другими строковыми значениями. Две строки сравниваются посимвольно до вы-

явления двух первых несовпадающих символов. Затем сравниваются (как обычные числовые значения) машинные коды этих несовпадающих символов и определяется наибольший код; соответствующая строка считается большей. Если первая строка короче второй и все ее символы совпадают с соответствующими символами второй строки, то большей считается вторая, более длинная строка. Две строки одинаковой длины с совпадающими в одинаковых позициях символами считаются равными; в частности, равными считаются две нулевые строки.

В случае числовых операндов выражения с несколькими операциями сравнения употребляются редко, а в случае строковых операндов их использование просто недопустимо. Это связано с тем, что выражения с операциями сравнения принимают целые значения (0 — «Ложь», —1 — «Истина»). Например, выражение A=B=C принимает значение «истина» (—1) только тогда, когда A=B и C=—1 или A<>B и C=0. Выражение типа A$=B$=C$ вызовет сообщение об ошибке "Type mismatch" («Несоответствие типов»), поскольку при выполнении первого сравнения A$=B$ получится числовое значение, которое нельзя сравнивать со строковой переменной C$.

Выражения с операциями сравнения наиболее часто используются при программировании в составе операторов условного перехода для задания условий, определяющих выбор очередного шага работы программы (гл. 9).

Логические выражения

Логические операции имеют смысл только для логических значений «Истина» и «Ложь». Поскольку эти значения являются результатами операций сравнения, неудивительно, что операндами логических выражений, как правило, являются выражения с операциями сравнения.

В Бэйсике ПВМ предусмотрено шесть логических операций: NOT, AND, OR, XOR, IMP и EQV. Каждая операция (кроме одноместной операции NOT) сопоставляет значениям двух своих операндов новое значение «Истина» или «Ложь», полученное в соответствии с правилами логики (табл. 8.2).

Обычно человеку легче разобраться в логическом выражении, если вместо абстрактных операндов в него входят знакомые повседневные понятия. Например, с помощью логических выражений можно определять свою позицию по отношению к кандидатам противоборствующих партий во время предвыборной кампании. В этом случае официальные позиции республиканской и демократической партий можно представить в качестве операндов. Для каждого пункта платформы первым операндом является утверждение: «Республиканцы поддерживают этот пункт», а вторым —

Таблица 8.2. **Результаты логических операций**

Значение первого операнда	Операция	Значение второго операнда	Значение выражения	Значение первого операнда	Операция	Значение второго операнда	Значение выражения
	NOT	И	Л	И	XOR	И	Л
	NOT	Л	И	И	XOR	Л	И
				Л	XOR	И	И
				Л	XOR	Л	Л
И	AND	И	И	И	EQV	И	И
И	AND	Л	Л	И	EQV	Л	Л
Л	AND	И	Л	Л	EQV	И	Л
Л	AND	Л	Л	Л	EQV	Л	И
И	OR	И	И	И	IMP	И	И
И	OR	Л	И	И	IMP	Л	Л
Л	OR	И	И	Л	IMP	И	И
Л	OR	Л	Л	Л	IMP	Л	И

«Демократы поддерживают этот пункт». Тогда с помощью логического выражения можно определить ложность или истинность утверждения «Я поддерживаю этот пункт».

Используя операцию NOT, операндом которой является позиция республиканцев, вы утверждаете, что ваши мнения противоположны. Выражение с операцией AND говорит о том, что вы поддерживаете пункт платформы только в том случае, если его поддерживают и республиканцы, и демократы. Выражение с операцией OR означает, что вы поддерживаете пункт платформы, если его поддерживает хотя бы одна из партий. В случае операции XOR вы поддерживаете пункт платформы, если его поддерживает только какая-нибудь одна из партий, но не обе. Операция EQV противоположна XOR: она говорит о том, что вы поддерживаете пункт платформы, если сразу обе партии поддерживают или отвергают его. Выражение с операцией IMP утверждает, что вы поддерживаете пункт платформы всегда, кроме случая, когда этот пункт поддерживают республиканцы и отвергают демократы.

Функции

Функции, в определенном смысле, можно рассматривать как понятие, промежуточное между командами и выражениями. Как и команде, каждой функции соответствует определенное командное слово, и, подобно выражению, она сопоставляет значениям нескольких операндов новое значение. В Бэйсике ПВМ имеется 61

стандартная функция, большинство из которых выдают в качестве результата числовое значение. Функции реализуют различные вычисления, анализ строковых данных, преобразование типов данных, а также позволяют осуществлять непосредственное управление вычислительной машиной и ее внешними устройствами и анализировать состояние и содержание файлов данных. Имеется 17 функций, результатами выполнения которых являются строковые значения, и еще 7 функций, которые хотя и вычисляют числовые значения, но связаны со строковыми данными. Все функции, входящие в Бэйсик ПВМ, перечислены в приложении А, а наиболее употребительные из них будут описаны ниже.

Функция может заменять некоторое выражение или быть его частью. Для использования функции следует указать ее идентифицирующее имя и перечислить за ним операнды, заключив их в скобки.

Пример. Строковая функция.

```
print string$(40,"*")
****************************************
Ok
```

Как видно из примера, функция STRING$ формирует строковое значение, состоящее из одного и того же повторяющегося символа. Функция имеет два операнда; значение второго операнда — повторяющийся символ, причем число его повторений (коэффициент повторения) задается первым операндом. Значение коэффициента повторения должно быть положительным и не больше 255.

Функция не обязательно должна иметь два операнда; в Бэйсике ПВМ имеются функции с одним операндом (их довольно много), а также несколько функций с тремя операндами. Примером функции с тремя операндами является функция INSTR, которая проверяет, является ли одна цепочка символов (строка) частью (подстрокой) некоторой другой. В качестве операндов этой функции выступает проверяемая строка, искомая строка и номер первой позиции, с которой нужно начинать просмотр проверяемой строки. В результате вычисления функции выдается целое число, равное номеру позиции начала вхождения искомой строки в проверяемую.

Пример

```
list
10 INPUT "Проверяемая строка: ", S$
20 INPUT "Искомая строка: ", F$
30 INPUT "Просмотр с позиции: ", P%
40 PRINT "Вхождение: ";INSTR(P%,S$, F$)
Ok
run
Проверяемая строка: Завораживающий
```

```
Искомая строка: щий
Просмотр с позиции: 7
Вхождение: 12
Ok
```

Первый операнд в INSTR (номер позиции начала просмотра) может отсутствовать; в этом случае считается, что он равен 1, т. е. проверяемая строка просматривается с первого символа.

Функции выделения подстроки

Наиболее часто используются следующие три функции выделения подстроки: LEFT$ (выделение начала строки), RIGHT$ (выделение конца строки) и MID$ (выделение середины строки). Для каждой из них надо указать исходную строку и число выделяемых символов, а в случае функции MID$ необходимо задать номер позиции в исходной строке, начиная с которой будет выделяться подстрока (слева направо).

Пример

```
list
10 INPUT "Строка: ", S$
20 INPUT "Количество символов: ",N%
30 INPUT "Номер нач. поз. для MID$: ",SP%
40 PRINT "Конец: ";RIGHT$(S$,N%)
50 PRINT "Начало: ";LEFT$(S$,N%)
60 PRINT "Середина: ";MID$(S$,SP%,N%)
Ok
run
Строка: Он рад своей победе
Количество символов: 6
Номер нач. поз. для MID$: 8
Конец: победе
Начало: Он рад
Середина: своей
Ok
```

Если количество выделяемых символов задать числом, большим длины всей строки, то ошибки не произойдет; в этом случае функции LEFT$ и RIGHT$ выдадут всю исходную строку, а MID$ — всю оставшуюся часть строки, начиная с заданной позиции. Когда число выделяемых символов задается равным нулю или номер начальной позиции для функции MID$ больше длины всей исходной строки, результатом выполнения любой из трех функций будет нулевая подстрока.

С помощью функции MID$ можно также присваивать значение некоторой подстроке:

```
MID$(S$,5,3) = "цемент"
```

В этом случае первый операнд определяет исходную строку, а второй операнд — номер позиции в этой строке, начиная с которой символы будут заменяться на новые значения. Третий операнд указывает, сколько символов в исходной строке получат новые значения; эти новые значения символов выбираются из строки, стоящей справа от знака равенства, начиная с ее первого символа. Третий операнд может отсутствовать, и тогда используется вся строка, стоящая справа от знака равенства.

Функции, определяемые пользователем

Помимо стандартных функций пользователь может определить и свои собственные. Так, можно определить функцию округления числовых значений до сотых долей (такая функция удобна, например, при выполнении операций над значениями денежных сумм):

```
10 DEF FNCENT(X)=INT(X*100+.5)/100
```

Оператор определения функции начинается с командного слова DEF, после которого должно быть указано имя функции, начинающееся обязательно с букв FN. Оставшаяся часть этого имени может представлять собой любое допустимое имя переменной (см. рис. 7.4 и табл. 7.2). Вслед за именем функции идет список *параметров*, заключенный в скобки. Параметры — это имена фиктивных переменных, резервирующих место в памяти для фактических значений, которые должны быть определены к моменту выполнения функции. В оставшейся части оператора записывается выражение, определяющее, какие действия производит данная функция. Операндами этого выражения обычно являются фиктивные переменные из списка параметров. В качестве операндов могут использоваться и другие переменные, элементы массивов, константы, функции. Разрешается использовать любые допустимые виды выражений при условии, что тип результирующего значения выражения совместим с типом, определяемым именем функции (и результат выражения, и значение функции должны быть либо оба строковыми, либо оба числовыми).

Действительное вычисление значения выражения происходит лишь в момент обращения к функции.

Пример

```
list
10  DEF FNCENT(X)=INT(X*100+.5)/100
100   INPUT "Объем реализации: $",AMT#
110   INPUT "Налоговая ставка:  ",RATE
120   TAX=ENCENT (RATE/100*AMT#)
500   PRINT ,, AMT #
510   PRINT,"Налог с оборота",TAX
520   PRINT, "Итого» ,FNCENT(TAX+AMT#)
Ok
```

```
run
Объем реализации: $9.97
Налоговая ставка: 6.5
                                    9.97
                  Налог с оборота   .65
                  Итого             10.62
Ok
```

Для использования в программе определенной ранее функции следует указать ее имя (начинающееся с FN), а затем в скобках перечислить значения, которые будут подставлены вместо фиктивных переменных в списке параметров функции. Каждое значение должно быть совместимо по типу с соответствующей фиктивной переменной (строковой или числовой). При выполнении функции указанные значения подставляются вместо фиктивных переменных только в этой функции и нигде больше, даже если в программе встречаются переменные с теми же именами, что и у фиктивных переменных. Если в выражение, определяющее функцию, входят какие-либо нефиктивные переменные, то при вычислении функции берутся их текущие значения. Тип и точность результирующего значения определяются именем функции.

Функции преобразования числовых данных

В Бэйсике ПВМ предусмотрен целый ряд функций, преобразующих данные из одного типа в другой. Из них по крайней мере три функции преобразуют значение с обычной или двойной точностью в целое число: функция FIX просто отбрасывает все цифры после десятичной запятой; функция INT определяет наибольшее целое, не превосходящее значения каждого ее операнда, а функция CINT округляет заданное число до ближайшего целого.

Пример. Различия между функциями FIX, INT и CINT.

```
list
10 INPUT "Число" ;A
20 PRINT "FIX(" ;A";") = ";FIX(A)
30 PRINT "INT(" ;A";") = ";INT(A)
40 PRINT "CINT(" ;A";") = ";CINT(A)
50 PRINT
60 RUN 'Рестарт; для останова нажать <Break>
Ok
run
Число? 101.625
FIX( 101.625 ) = 101
INT( 101.625 ) = 101
CINT( 101.625 ) = 102
```

```
Число? 500.1
FIX( 500.1 ) = 500
INT( 500.1 ) = 500
CINT( 500.1 ) = 500

Число? —265.1
FIX( —265.1 ) = —265
INT( —265.1 ) = —266
CINT( —265.1 ) = —265

Число? —133.9
FIX( —133.9 ) = —133
INT( —133.9 ) = —134
CINT( —133.9 ) = —134

Число?
Break in 10
Ok
```

Результат вычисления как функции FIX, так и функции INT является значением с обычной точностью (с нулевой дробной частью). Значение, полученное в результате вычисления CINT, является значением целого типа и должно изменяться в пределах от —32768 до 32767. Поэтому операнды этой функции не должны принимать значений, выходящих за указанные пределы; в противном случае произойдет ошибка.

Функция CSNG округляет заданное значение с двойной точностью до семи или менее значащих цифр следующим образом:

```
print csng(97.5436750000001)
97.54368
Ok
```

Функция CDBL преобразует целое значение или значение с обычной точностью в значение с двойной точностью (фактическая точность самого числа при этом не повышается).

Пример

```
print cdbl(10%/3%)
3.333333253860474
Ok
print cdbl(1.1!/1.5!)
.7333333492279053
Ok
```

Из данного примера следует, что получаемые значения с двойной точностью правильны лишь до седьмого знака, а все следующие за этим знаком цифры могут оказаться неверными.

Цифровые строки

Кроме трех числовых форматов числа можно представить еще и в виде строковых значений, например «1234.56». В ряде случаев удоб-

нее работать с числовыми значениями, а не с их представлениями в виде цифровых строк. Так, употребление цифровых строк в выражениях с операциями сравнения может привести к неверным результатам. Например, выражение "10"="10.0" ложно, хотя 10=10.0 истинно.

Цифровую строку можно преобразовать в числовое значение с помощью функции VAL. Преобразование начинается с крайнего левого знака и заканчивается, когда преобразован последний знак либо когда встретился недопустимый нецифровой знак. Цифровая строка может содержать любое количество стоящих впереди пробелов, знак плюс или минус, любое число цифр и десятичную точку; допустима также запись числа в экспоненциальном представлении. Если строка начинается с нецифрового знака, то результатом выполнения VAL является нуль.

Примеры. Действие функции VAL.

```
print val("1234.56789");val("—1234.56789!")
  1234.56789 —1234.568
Ok
print val ("9.29558d7 миль"),val("Индекс 94596")
  92955800        0
val("47.5%")
Syntax error
Ok
```

Точность выполняемого функцией VAL преобразования определяется «точностью» цифровой строки — количеством цифр и абсолютной величиной числа. Если первым нецифровым символом является восклицательный знак, то результат преобразования представляет собой число с обычной точностью. В то же время знаки # и % не интерпретируются здесь, как в обычных числовых константах. Так, если первым нецифровым символом является знак процента (%), это будет считаться ошибкой.

Действие функции STR$ противоположно действию функции VAL: она преобразует числовое значение в цифровую строку. Преобразованное в строку числовое значение будет иметь тот же формат, что и при выводе с помощью оператора PRINT. Единственное отличие от оператора PRINT — отсутствие в данном случае дополнительного пробела после последней цифры строки.

Пример

```
print 6620; "км/ч" ,str$(6620); "км/ч"
  6620 км/ч 6620 км/ч
Ok
print str$(&h3f)
  63
Ok
```

```
print str$(.0000000299792458);" м/с"
.0000000299792458 м/с
Ok
```

Кроме STR$ имеется еще пять функций, преобразующих различными способами числовое значение в цифровую строку. Функции OCT$ и HEX$ преобразуют десятичное числовое значение в его восьмеричное и соответственно шестнадцатеричное представления в виде цифровых строк (при необходимости производится округление). Преобразуемое значение должно быть не меньше —32768 и не больше 65535.

Примеры

```
list
10 INPUT "Число: " ,N%
20 PRINT N%;"есть ";HEX$(N%);" шестн., ";
   OCT$ (N%);" восьм."
30 RUN 'Рестарт; для останова нажать <Break>
Ok
run
Число: 100
  100 есть 64 шестн., 144 восьм.
Число: 129,8
  129 есть 82 шестн., 202 восьм.
Число:
Break in 10
Ok
```

Функции MKI$, MKS$ и MKD$ предназначены для работы с дисковыми файлами данных; они преобразуют числовые величины в целые значения и в числа с обычной и двойной точностью, представленные в виде цифровых строк (гл. 12).

Функции преобразования символов в код ASCII

Функция ASC позволяет получать для произвольного символа значение его числового кода ASCII (коды всех символов приведены в приложении D). Обратная ей функция CHR$ интерпретирует произвольное числовое значение как код и выдает соответствующий этому коду символ.

Пример. Использование функций ASCII и CHR$ в режиме немедленной обработки.

```
print asc( "A" )
  65
Ok
print chr$(64+33)
a
Ok
```

Глава 9

ОРГАНИЗАЦИЯ ПРОГРАММЫ

Лишь немногие программы могут быть записаны в виде длинной последовательности выполняемых один за другим операторов. В данной главе описываются средства Бэйсика, позволяющие управлять этим процессом. Они включают в себя оператор безусловного перехода, операторы условного перехода и операторы цикла. Кроме того, рассматриваются способы структурирования программ путем создания программных модулей и организации их совместной работы. Использование этих способов существенно облегчает разработку и отладку программ и повышает эффективность последних.

Организация ветвления

Естественный порядок выполнения операторов означает, что операторы выполняются один за другим в порядке их написания. Для изменения этого стандартного порядка выполнения операторов в Бэйсике предусмотрено несколько средств.

Оператор GOTO

С помощью оператора GOTO можно изменять естественный порядок выполнения операторов и передавать управление любой заранее заданной строке программы, которая указывается после командного слова GOTO.

Пример

```
list
10 INPUT "Тип издания: " ,ITEM$
20 GOTO 100
30 ITEM$="Тип издания — "+ITEM$
100 PRINT ITEM$
Ok
run
Тип издания: ежемесячник
Ежемесячник
Ok
```

Номер строки задается в GOTO в виде константы. Если в константе встречается десятичная точка, то интерпретатор Бэйсика игнорирует ее и все следующие за ней десятичные цифры.

◆◆◆ Не разрешается использовать в качестве номера строки число в экспоненциальном представлении, переменную или выражение.

Если в программе нет строки, номер которой указан в GOTO, то выдается сообщение "Undefined line number ..." («Неопределенный номер строки...»).

В операторе GOTO может быть указан номер любой строки программы, в том числе и строки, в которой нет ничего кроме комментария. Поскольку, однако, оператор REM не исполняется, лучше передавать управление следующему за ним исполняемому оператору. В этом случае программа будет выполняться правильно и после удаления комментариев. Следующий пример иллюстрирует, что может произойти, если не придерживаться этой рекомендации.

Пример

```
list
100 REM Инициализация массивов
110 READ COST(N)
120 READ PRICE(N)
130 N=N+1:GOTO 100
9000 DATA 375,44.50,90
Ok
delete 100
Ok
run
Undefined line number in 130
Ok
```

В приведенном примере сообщение об ошибке не появится, если оператор GOTO в строке 130 заменить на оператор GOTO 110.

Выполнение оператора GOTO в режиме немедленной обработки аналогично выполнению команды RUN, но в отличие от этой команды оператор GOTO просто начинает выполнение программы с первого оператора указанной строки, не присваивая никаких начальных значений переменным и не меняя никаких данных. С этой точки зрения оператор GOTO в режиме немедленной обработки полезен при отладке и тестировании. В остальных случаях использовать его в этом режиме не рекомендуется, поскольку существует большая вероятность задать неправильный номер строки или непреднамеренно изменить значение переменной так, что фактически изменится вся программа, и это приведет к появлению многочисленных ошибок.

Оператор ON-GOTO

Часто возникает необходимость передавать управление не одной и той же программной строке, а разным строкам — в зависимости от сложившейся в ходе выполнения программы ситуации. Средством такой передачи управления является оператор ON-GOTO, ко-

торый содержит выражение и список номеров строк. Значение выражения рассматривается как указатель этого списка: 1 означает, что переход должен осуществляться к строке, номер которой указан в списке первым, 2 означает, что переход должен осуществляться к строке, номер которой указан вторым, и т. д.

Пример

```
list
1000 INPUT "Введите номер элемента
     (от 1 до 3): ",N%
1010 ON N GOTO 1100,1200,1300
1020 PRINT "Номер не определен":GOTO 1000
1100 PRINT "Водород":GOTO 1000
1200 PRINT "Гелий":GOTO 1000
1300 PRINT "Литий":GOTO 1000
Ok
run
Введите номер элемента (от 1 до 3): 2
Гелий
Введите номер элемента (от 1 до 3): _
```

Дробные значения выражения, указанного в операторе ON-GOTO, округляются до ближайшего целого числа. Если значение этого выражения равно 0 или больше числа элементов списка номеров, то оператор ON-GOTO не выполняется и управление передается следующему за ним оператору. При отрицательном значении выражения выдается сообщение «Illegal function call...» («Недопустимый вызов функции...»).

Условная передача управления

С помощью оператора ON-GOTO при некоторых дополнительных действиях можно осуществить практически любое управление порядком выполнения операторов Бэйсик-программы. Однако в ряде случаев этот оператор неудобен и приводит к слишком громоздким конструкциям. В подобных ситуациях лучше использовать другой оператор условного перехода — оператор IF-THEN.

Оператор IF-THEN

Оператор IF-THEN предписывает выполнять некоторое действие только в том случае, когда выполняется заданное условие. Это условие записывается в операторе IF-THEN в виде логического выражения, а действие, которое нужно выполнить, задается с помощью обычных операторов Бэйсика. Если выражение принимает значение «Истина», то действие, заданное оператором, выполняется.

В противном случае, т. е. когда выражение «Ложно», оператор, задающий действие, пропускается.

Пример. Использование оператора IF-THEN.

```
list
100 INPUT   "Введите два числа: ",A,B
110 IF A>B  THEN PRINT "1-е число больше"
120 IF B>A  THEN PRINT "2-е число больше"
130 IF B=A  THEN PRINT "Числа равны"
139 ' Если оба числа не равны 0, то рестарт
140 IF A<>0 AND B<> 0 THEN GOTO 100
150 END
Ok
run
Введите два числа: —3,5
2-е число больше
Введите два числа: 1e4,1000
Числа равны
Введите два числа: 0,0
Числа равны
Ok
```

Если действие в IF-THEN задается оператором GOTO (как в в строке 140 приведенной выше программы), то можно опустить либо командное слово THEN, либо командное слово GOTO, но не оба сразу. Таким образом, оператор в строке 140 приведенного выше примера можно записать в виде

```
140 IF A<>0 AND B<>0 THEN 100
```

либо

```
140 IF A<>0 AND B<>0 GOTO 100
```

Действие в операторе IF-THEN можно задавать последовательностью нескольких операторов, разделенных двоеточием.

Пример

```
list
90 GOTO 1000
900  PRINT "Ат. вес элемента ";ELMT$; "равен";AW
1000 INPUT "Введите номер элемента: ",N%
1010 IF N%=0 THEN END
1100 IF N%=18 THEN ELMT$="Аргон"
     :AW=39.948:GOTO 900
1200 IF N%=26 THEN ELMT$="Железо"
     :AW=55.847:GOTO 900
1300 IF N%=82 THEN ELMT$="Свинец"
     :AW=207.2:GOTO 900
1400 PRINT "Атомный вес элемента";
```

```
    N%; "неизвестен"
1410 GOTO 1000
Ok
run
Введите номер элемента: 26
Ат. вес элемента Железо равен 55.847
Введите номер элемента: _
```

В последовательность операторов, определяющих конструкцию IF-THEN, может входить любой оператор; если при этом используется оператор GOTO, то он, естественно, будет последним реально выполняемым.

◆◆◆ Следует соблюдать осторожность и при использовании оператора REM: поскольку интерпретатор игнорирует всю часть строки, расположенную после командного слова REM, комментарии нужно помещать в самом конце строки.

Конструкция IF-THEN-ELSE

Обычно оператор IF-THEN разветвляет программу на 2 части, которые затем соединяются вновь. Одна ветвь выполняется, если логическое выражение оператора IF-THEN истинно, а другая ветвь, если оно ложно. Обе ветви по окончании работы передают управление, как правило, одному и тому же оператору программы.

Пример

```
list 940-960
940 IF TERMS$= "S" THEN DSCNT=40:GOTO 960
950 DSCNT=30
960 PRINT STR$(DSCNT); "% NET PRICE $";
    FNCENT(TOTAL*(1—DSCNT/100))
```

Операторы, выполняемые когда выражение в IF-THEN ложно, могут располагаться в отдельной строке программы (как в приведенном примере), а могут включаться в сам оператор IF-THEN. В последнем случае они записываются в конце этого оператора после командного слова ELSE.

Пример

```
list 940-960
940 IF TERMS$ = "S" THEN DSCNT=40 ELSE
    DSCNT=30
960 PRINT STR$(SDCNT); "% NET PRICE $";
    FNCENT (TOTAL*(1—DSCNT/100))
```

◆◆◆ Заметим, что перед командным словом ELSE двоеточие не ставится.

Циклы

Часто возникает необходимость выполнять одни и те же шаги программы по нескольку раз. В подобной ситуации многократного переписывания одних и тех же строк программы можно в нужном месте передать управление предыдущим операторам с тем, чтобы они выполнялись повторно. Однако в некоторых случаях требуется повторять не всю программу, а лишь некоторую ее часть и не бесконечно долго, а вполне определенное число раз. Для этого необходимы средства, позволяющие определять, в какой момент следует прекратить повторное выполнение выделенной части программы. Можно, например, считать число повторений (или итераций), используя целую переменную, и в зависимости от ее значения с помощью оператора IF-THEN определять каждый раз, следует ли прекратить повторения или нет.

Пример

```
list
30 CTR%=1 'Начальное значение счетчика
40 IF CTR%>8 THEN 70
50 PRINT 2^CTR%;
60 CTR%=CTR%+1:GOTO 40
70 REM Возобновление последовательности
80 END
Ok
run
2 4 8 16 32 64 128 256
Ok
```

При выполнении этой программы строки 50 и 60 повторяются до тех пор, пока значение переменной CTR% не превысит 8; при каждом выполнении строки 60 значение этой переменной увеличивается на 1. Поэтому передача управления снова на строку 50 будет повторяться ровно 8 раз.

Операторы FOR и NEXT

Как было показано в предыдущем примере, циклы в программе можно организовывать с помощью обычных операторов условного перехода. Однако при этом приходится использовать несколько различных операторов: оператор присваивания счетчику начального значения, оператор IF-THEN, содержащий условия окончания цикла, и оператор, увеличивающий значение счетчика. Для упрощения программирования циклов в Бэйсике ПВМ предусмотрены специальные операторы FOR и NEXT, включающие все необходимые действия по организации цикла.

Пример

```
list
40 FOR CTR%=1 TO 8
50 PRINT 2^CTR%;
60 NEXT CTR%
70 REM Возобновление последовательности
80 END
Ok
run
2 4 8 16 32 64 128 256
Ok
```

Оператор FOR идентифицирует начало циклического участка программы, дает имя числовой переменной, которая будет служить счетчиком числа повторений цикла, присваивает этому счетчику начальное значение и устанавливает максимально возможное значение числа повторений. Счетчик должен быть простой переменной и не может быть элементом массива. При выполнении оператора FOR проверяется текущее значение счетчика циклов; если оно не превосходит максимального, то выполняются операторы программы, расположенные между FOR и NEXT. При выполнении оператора NEXT происходит передача управления на начало цикла — на оператор, следующий за FOR.

Операторы FOR и NEXT всегда должны идти в паре и быть согласованы друг с другом. Если оператор NEXT стоит перед оператором FOR с тем же счетчиком циклов, то при выполнении такого NEXT выдается сообщение об ошибке "NEXT without FOR..." («Нет оператора FOR для данного NEXT...»). Подобное сообщение выдается и в том случае, когда операторов NEXT в программе недостаточно для того, чтобы каждому FOR соответствовал свой NEXT.

В тот момент, когда при выполнении оператора FOR обнаружено, что текущее значение счетчика превосходит максимальное значение (установленное в операторе FOR), происходит передача управления оператору, следующему за NEXT. Таким образом, когда значение счетчика превосходит максимальное, входящие в цикл операторы не выполняются.

Пример

```
list
10 FOR K=10 TO 9
20 PRINT "Цикл выполнен"
30 NEXT K
40  PRINT "Конечное значение =" ;K
Ok
```

```
run
Конечное значение = 10
Ok
```

Шаг цикла FOR/NEXT

Обычно после каждого очередного выполнения всех операторов, образующих цикл, содержимое его счетчиков увеличиваетя на 1. В ряде случаев, однако, оказывается удобнее, чтобы величина приращения была отлична от 1. Это можно сделать, добавив в конце оператора FOR ключевое слово STEP и указав за ним нужное значение, на которое будет увеличиваться счетчик на каждой итерации цикла. В качестве величины приращения, или шага цикла, разрешается использовать любые значения, в том числе отрицательные и дробные.

Пример. Использование отрицательного дробного значения «приращения» (в действительности показание счетчика каждый раз уменьшается).

```
list
40 FOR L=3.5 TO 1 STEP —.5
50 PRINT L;
60 NEXT L
70 REM Возобновление последовательности
80 END
Ok
run
3.5 3 2.5 2 1.5 1
Ok
```

Если шаг цикла отрицателен, то цикл повторяется до тех пор, пока значение управляющей переменной не станет *меньше* конечного значения, заданного в операторе FOR. В этом случае начальное значение счетчика в операторе FOR должно быть больше конечного.

Использование счетчика цикла FOR/NEXT

Переменная, играющая роль счетчика повторений (заданная в операторе FOR), может использоваться внутри цикла точно так же, как любая другая числовая переменная. В частности, можно даже изменять ее значение, что, однако, делать не рекомендуется, так как это может повлиять на число повторений операторов цикла: оператор NEXT всегда дает приращение самому последнему значению счетчика.

При переходе из цикла на ветвь по какому-либо оператору условного или безусловного перехода значение счетчика повторений

выбирается равным его последнему значению внутри цикла. При нормальном завершении цикла, т. е. при выходе из цикла через оператор NEXT, значение счетчика равно последнему его значению плюс величина шага цикла. В последнем примере переменная L при выполнении строки 70 будет иметь значение 0,5.

Задание параметров и организация вложенных циклов FOR/NEXT

Для задания начального и конечного значений счетчика повторений цикла, а также величины приращения для счетчика в операторе FOR можно использовать константы, переменные или выражения.

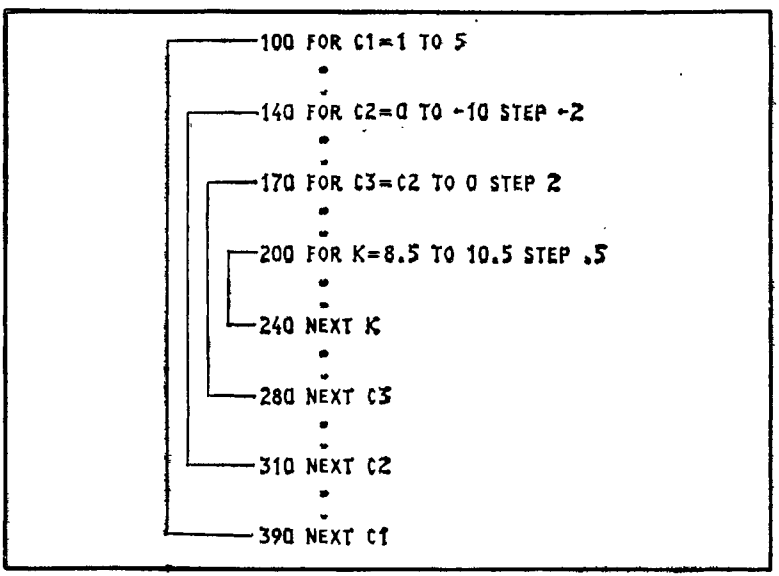


Рис. 9.1. Вложенные циклы FOR/NEXT.

Но как только оператор FOR выполнен, эти значения считаются неизменными в течение всего времени выполнения итераций цикла. Изменение значений переменных внутри цикла не влияет на начальное, конечное значения счетчика и на значение шага, которые были вычислены в начале цикла.

Циклы FOR/NEXT могут быть вложены друг в друга (рис. 9.1). Для этого достаточно опустить имя переменной в операторе NEXT, и тогда оператор NEXT будет соответствовать последнему выполненному оператору FOR. Однако так поступать не следует, если во вложенных циклах возможна передача управления из внутреннего цикла во внешний, поскольку в этом случае указанное выше соот-

ветствие будет нарушено и при выполнении программы возникнет ошибка.

Пример

```
list
10 DIM N$(2,10)
20 FOR J%=1 TO 2
30 FOR K%=1 TO 10
40 INPUT "Имя: ",N$(J%,K%)
50 IF N$(J%,K%)= "" THEN GOTO 70
60 NEXT
70 NEXT
Ok
run
Имя: Милтон
Имя: Чосер
Имя:
NEXT without FOR in 70
Ok
```

В приведенной программе оператор NEXT в строке 60 является окончанием внутреннего цикла, начинающегося в строке 30. Однако если в строке 50 произойдет передача управления на строку 70, то оператор NEXT строки 70 будет воспринят как последний оператор внутреннего цикла. Но строка 70 формально является последней строкой внешнего цикла, начинающегося в строке 20. Поскольку при указанной передаче управления строка 70 интерпретируется как завершающая внутренний цикл, для внешнего цикла не найдется завершающей строки с оператором NEXT. В результате программа остановится и будет выдано сообщение об ошибке.

Операторы WHILE и WEND

Программировать цикл с помощью операторов FOR и NEXT удобно, если некоторые действия нужно выполнять известное, заранее заданное число раз. Однако иногда возникает необходимость изменять число повторений в процессе выполнения цикла либо прерывать процесс выполнения цикла в зависимости от сложившейся ситуации. В подобных случаях в FOR/NEXT-цикл можно ввести оператор IF-THEN, который при выполнении определенных условий передаст управление за пределы цикла. Можно использовать также операторы WHILE и WEND, предназначенные специально для организации циклов с нестандартными, нерегулярными, непредсказуемыми условиями завершения и позволяющие обходиться без дополнительных операторов IF-THEN.

Оператор WHILE, с которого начинается в программе цикл WHILE/WEND, состоит из командного слова WHILE и выражения,

заданного с помощью отношений и логических операций. Когда выражение принимает истинное значение, выполняются операторы, следующие за оператором WHILE вплоть до оператора WEND. Как только встречается оператор WEND, управление передается на оператор WHILE. При повторном выполнении оператора WHILE заново вычисляется содержащееся в нем выражение и, если оно по-прежнему истинно, снова выполняются операторы цикла (расположенные между WHILE и WEND). Как только выражение в операторе WHILE будет иметь ложное значение, управление передается оператору, следующему за WEND.

Пример

```
list
1000 INPUT "Строка для сжатия: ",S$
1010 RMV$= " " ' Удаление указанных символов из введенной
     строки
2000 WHILE INSTR(1,S$, RMV$)<>0
2009 '— — Удаление символов по одному — — — —
2010 S$=LEFT$(S$,INSTR(1,S$,RMV$) — 1)+
   RIGHT$(S$,LEN(S$)—INSTR(1,S$,RMV$))
2020 WEND
3000 PRINT S$
ok
run
Строка для сжатия: а б в г д е
абвгде
Ok
```

Приведенная программа сжимает исходную строку, удаляя из нее все пробелы. Пока пробелы присутствуют в строке, выражение в операторе WHILE (строка 2000) принимает истинное значение и выполнение цикла WHILE/WEND (строки 2000—2020) повторяется. На каждой итерации цикла из сжимаемой строки удаляется один пробел (строка 2010). Как только все пробелы будут удалены, операторы цикла больше не выполняются, а управление передается следующему за WEND оператору, который выводит сжатую строку (строка 3000). При сжатии можно удалять вместо пробела любой другой символ; для этого достаточно изменить соответствующим образом значение переменной RMV$ (строка 1010).

В операторе WHILE могут также использоваться арифметические выражения. В этом случае выполнение цикла повторяется до тех пор, пока значение такого выражения отлично от нуля. Например, вместо оператора WHILE в приведенной выше программе можно было бы использовать следующий оператор:

```
2000 WHILE INSTR (1,S$,RMV$)
```

Допускаются вложенные друг в друга циклы WHILE/WEND, при этом каждому оператору WHILE должен соответствовать свой оператор WEND, иначе произойдет ошибка.

Структурирование программ

Чем сложнее и длиннее программа, тем острее ощущается потребность в специальных методах разработки и организации ее функционирования. Трудности, обусловленные бессистемным написанием большой и сложной программы, можно сравнить с трудностями, возникающими при попытке съесть сразу целиком весь батон колбасы; в то же время, если разрезать колбасу на ломтики, то съесть ее не представит никакого труда.

Простейший способ разбиения программы на части — это выделение в ней обособленных разделов по их функциональному назначению. Обычно первую часть программы составляют различные описания и определения, вторую часть — процедуры инициализации, третью — ввод данных, затем следует обработка и, наконец, вывод данных. Для того чтобы в целостной программе отделить один раздел от другого, можно использовать в каждом из них свой диапазон номеров строк, достаточно удаленный от диапазонов других разделов. Если при написании программы установлен какой-либо стандартный способ нумерации строк для различных ее разделов, то впоследствии можно легко находить строки с операторами DIM, с операторами DEF, с операторами DATA и т. д.

Некоторые программные разделы можно в свою очередь подразделять на модули, модули разбивать на отдельные сегменты и т. д. Конечная цель состоит в том, чтобы полученные в результате окончательного разбиения элементарные модули были достаточно малы и их можно было бы легко разрабатывать, программировать и отлаживать. Дополнительное преимущество такого подхода к созданию больших сложных программ состоит в том, что многие из полученных модулей можно будет использовать как некие «суперкоманды» в любой другой программе.

Подпрограммы

Часто в различных местах одной и той же программы должна выполняться одна и та же процедура, например сжатие строк. Для этого при написании программы можно включить в нее (в соответствующих местах) несколько экземпляров цикла WHILE/WEND из предыдущего примера, однако такое дублирование неэкономно. Вместо этого можно выделить из основной программы процедуру сжатия строк и при необходимости передавать ей управление из основной программы. Такая процедура называется *подпрограммой*, а передача ей управления — *вызовом подпрограммы*. После того

как подпрограмма закончила работу, управление передается в то место основной программы, откуда произошло обращение к подпрограмме.

Операторы GOSUB и RETURN

Оператор GOSUB осуществляет вызов подпрограммы. Подобно оператору GOTO, он передает управление определенной указанной в нем строке, но при этом еще запоминается и точка вызова, в которую необходимо вернуться после завершения работы вызванной подпрограммы. Завершает работу подпрограммы оператор RETURN. При его выполнении управление передается в основную программу — оператору, следующему за GOSUB, инициировавшим вызов подпрограммы.

Пример. Использование подпрограмм.

```
list
1000 INPUT "Строка для сжатия: ",FS$
1009 'Удаление пробелов из FS$
1010 S$=FS$:RMV$=" ":GOSUB 2000
1030 PRINT S$
1040 END
1997 '==Подпрограмма сжатия строк======
1998 ' Значение RMV$ — удаляемый символ
1999 В S$ — исходная строка и результат
2000 'WHILE INSTR(1,S$,RMV$)<>0
2010 S$=LEFT$(S$,INSTR (1,S$,RMV$)—1)+
    RIGHT$(S$LEN(S$)—INSTR(1,S$,RMV$))
2020 WEND
2030 RETURN
Ok
```

В приведенной программе для удаления всех пробелов из строкового значения используется соответствующая подпрограмма. Эта подпрограмма (строки 2000—2020) исключает из строки, являющейся значением переменной S$, все символы, равные значению переменной RMV$. Поэтому в основной программе перед вызовом подпрограммы надо присвоить этим переменным соответствующие значения (строка 1010). Оператор GOSUB (строка 1010) запускает подпрограмму, а после ее завершения оператор RETURN (строка 2030) осуществляет возврат управления оператору, следующему за GOSUB; в данном случае это оператор PRINT (строка 1030). Оператор END (строка 1040) отделяет основную программу от подпрограммы.

Обращение к подпрограмме может происходить из нескольких мест основной программы. Например, если в программу предыдущего примера добавить строку, приведенную ниже, то подпрограм-

ма будет вызываться еще раз для удаления знака переноса из строки, являющейся значением переменной S$:

```
1020 RMV$="-":GOSUB 2000 'Удал. переносов
```

Подпрограммы и стек адресов возврата

При многократном выполнении выхода из подпрограмм с помощью операторов GOTO, ON-GOTO или IF-THEN (вместо оператора RETURN) в конце концов может появиться сообщение об ошибке «Out of memory ...» («Нехватка памяти...»). Это следует иметь в виду при использовании внутри подпрограммы операторов передачи управления за ее пределы. Причина возникающей ошибки довольно своеобразна: дело в том, что Бэйсик-интерпретатор отводит сравнительно небольшой участок динамической памяти для хранения списка адресов возврата из подпрограмм. Такой список организуется в виде стека и называется *рабочим стеком*. Каждый раз при вызове подпрограммы в стек заносится соответствующий этому вызову адрес возврата. При выполнении оператора RETURN из стека выбирается адрес возврата, записанный последним, и происходит передача управления по этому адресу, после чего он удаляется из стека. Операторы, подобные GOTO или IF-THEN, осуществляют выход из подпрограммы, не затрагивая стека. Если такой выход из подпрограмм происходит часто, то адреса из стека не удаляются, а лишь добавляются при каждом новом вызове подпрограммы, так что в результате размер стека может превысить величину отведенного ему участка памяти. В этом случае программа прекращает работу и выдается сообщение об ошибке (гл. 15).

Для нестандартного выхода из подпрограммы на конкретную строку программы в расширенном Бэйсике предусмотрена возможность задавать в операторе RETURN номер строки. При выполнении такого оператора RETURN управление передается строке, номер которой указан в операторе RETURN, после чего из стека удаляется последний занесенный в него адрес. Описанное средство следует использовать с осторожностью, поскольку при выполнении процедур в середине какого-либо цикла FOR/NEXT или WHILE/WEND эти циклы будут оставаться в активном состоянии.

Оператор ON-GOSUB

Оператор ON-GOSUB, как и обычный оператор GOSUB, осуществляет вызов подпрограммы, но при этом управление передается строке, номер которой выбирается из заданного в ON-GOSUB списка номеров в соответствии со значением указанного в ON-GOSUB выражения. В этом отношении оператор ON-GOSUB аналогичен оператору ON-GOTO.

Пример

```
890 ON ASC(R$)—48 GOSUB 910, 930, 950, 970
```

Значение выражения служит индексом в списке номеров: значение, равное 1, означает, что управление передается строке с номером, стоящим первым в списке номеров; значение, равное 2, означает, что управление передается строке с номером, стоящим вторым в этом списке, и т. д. Если значение выражения является дробным, то оно округляется до ближайшего целого. При значении выражения, равном 0 или большем числа элементов списка номеров, никакой передачи управления по оператору ON-GOSUB не происходит, и программа продолжает выполняться со следующего за ON-GOSUB оператора. При отрицательном значении выражения выдается сообщение об ошибке «Illegal function call ...» («Недопустимый вызов функции...»).

Вложенные подпрограммы

Обращения к подпрограммам могут вкладываться одно в другое; при этом вложенность подпрограмм несколько отличается от вложенности циклов или вложенности скобок в выражении. В случае подпрограмм вкладываются не сами подпрограммы, а вызывающие их операторы GOSUB. Таким образом, первая подпрограмма может вызывать вторую, вторая — третью и т. д.

Пример

```
list
1100 GOSUB 8000 'Вывод цитаты в рамке
2000 END
7999 '==Подпрограмма печати цитаты=======
8000 GOSUB 9000 'Печать строки звездочек
8010 PRINT "Если программа неправильна, не имеет
     значения, какова ее эффективность."
8020 PRINT "Обычно важнее удобочитаемость программы, чем ее
     эффективность."
8030 GOSUB 9000 'Печать строки звездочек
8040 RETURN
8999 '==Подпрограмма печати звездочек==
9000 FOR XCOUNT%=1 TO 20
9010 PRINT "* ";
9020 NEXT XCOUNT%
9030 RETURN
Ok
```

Основная программа в этом примере состоит лишь из оператора GOSUB (строка 1100) и оператора END (строка 2000). Первая под-

программа (строки 8000—8040) осуществляет печать текста цитаты, окаймленного звездочками. Для печати звездочек первая подпрограмма вызывает вторую (строки 9000—9030). Окончательный результат выполнения всей программы выглядит следующим образом:

```
run
* * * * * * * * * * * * * * * * * * * * * * *
   Если программа неправильна, не имеет
значения, какова ее эффективность.
   Обычно важнее удобочитаемость программы,
чем ее эффективность.
* * * * * * * * * * * * * * * * * * * * * * *
Ok
```

Вложенные подпрограммы являются довольно мощным средством разработки и отладки больших программ. Используя принцип вложений, можно не только разбивать сложную программу на отдельные модули и для каждого из них писать свою подпрограмму, но и разделять на модули любые подпрограммы. Чем меньше будет каждый выделенный программный модуль, тем легче будет его программировать и отлаживать и тем больше вероятность его многократного использования в различных других программах.

Рекурсия

Вид вложенности, когда подпрограмма обращается к самой себе, называется *рекурсией*. Понятие рекурсии охватывает также и ситуацию, при которой первая подпрограмма вызывает вторую, а та в свою очередь вызывает первую.

Пример. Использование рекурсии.

```
list
1000 INPUT "Строка для сжатия: ",FS$
1009 '— —Удаление пробелов из FS$
1010 S$=FS$:RMV$=" ":GOSUB 2000
1030 PRINT S$
1040 END
1999 '==Рекурсивная подпрограмма сжатия строк=
1998 ' B RMV$ — удаляемые символы
1999 ' B S$ — исходная строка и результат
2000 IF INSTR(1,S$,RMV$)<>0
     THEN S$=LEFT$(S$,INSTR(1,S$,RMV$)—1)+
     RIGHT$(S$,LEN(S$)—INSTR(1,S$,RMV$)):
     GOSUB 20000
2010 RETURN
Ok
```

Эта программа удаляет пробелы из заданного строкового значения точно так же, как это делалось в одном из ранее рассмотренных

в этой главе примеров. До тех пор пока подпрограмма (строки 2000 и 2010) находит пробелы в переменной S$, она продолжает вызывать саму себя. При каждом обращении к подпрограмме исключается один пробел: после удаления всех пробелов управление передается последовательно по всем адресам возврата, которые заносились в стек при каждом вызове до тех пор, пока не произойдет передача управления по адресу, соответствующему первому вызову, т. е. пока не будет осуществлен возврат в основную программу (строка 1010).

Существует ограничение на число обращений подпрограммы к себе самой. Оно зависит от нескольких факторов, в частности от используемой версии Бэйсика, количества других вложенных одна в другую подпрограмм, выполняемых в этот же момент времени, и от размера рабочего стека. Результаты выполненения тестовых программ показывают, что при стандартном размере этого стека максимальный уровень вложенности для кассетного Бэйсика и дискового Бэйсика равен 44, а для расширенного Бэйсика — 33.

Оверлейная структура программы

Объем программ колеблется от единиц до сотен строк, и в них могут использоваться как несколько простых переменных, так и множество больших массивов. С некоторого момента размер программы начинает превышать объем динамической памяти. Такую программу необходимо разбивать на сегменты. Каждый сегмент в этом случае хранится на диске как самостоятельная программа, но выполняет лишь часть общей задачи. Когда текущий сегмент завершает работу, он инициирует загрузку очередного программного сегмента, предназначенного для выполнения следующего шага задачи, и передает управление этому новому сегменту. Описанная структура программы называетя *оверлейной*, а отдельный программный сегмент — *оверлеем* (от английского слова overlay — перекрытие, наложение; каждый вновь поступающий сегмент как бы накладывается на сегмент, находящийся в данный момент в памяти).

Для соединения каждого очередного сегмента со следующим лучше всего использовать оператор CHAIN, имеющий вид

```
200 CHAIN "prog2"
```

Строковое значение, указанное в конце оператора CHAIN, является именем некоторой Бэйсик-программы. В общем случае помимо имени самой программы оно должно содержать еще и имя файла, в котором она хранится. Если имя файла опущено, как в приведенном выше примере, то по умолчанию считается, что программа с заданным именем находится в файле с расширением имени BAS.

Для того чтобы показать, как осуществляется сцепление программных сегментов, рассмотрим программу, которая выдает названия

чисел на разных языках. Сначала эта программа по требованию пользователя формирует названия чисел на одном языке, а затем, как только пользователь просит выдать название числа, большего, чем предусмотренные в программе, происходит переключение на другой язык.

Пример. Программа, которая позволяет получить название любого числа от нуля до девяти на испанском языке.

```
list
10 DIM W$(9)
20 FOR XC%=0 TO 9:READ W$(XC%):NEXT 'Ввод названий
   чисел
30 READ LANG$, CUE$ 'Ввод названия языка и запросного
   слова
50  PRINT CUE$;LANG$; " : "
100  PRINT CUE$;:INPUT N% 'Запрос числа, название которого
     нужно выдать
110  IF N%>9 THEN 200 'Перейти к другому языку, если
     запрошенное число больше 9
120  PRINT N%;LANG$;" ";W$(N%)  'Вывод названия числа
130  GOTO 100
200  CHAIN "prog 2" 'Загрузка другого языка
6990 '——Таблица названий чисел от 0 до 9————
7000 DATA zero,uno,dos,tres,cuatro,cinco, seis,siete,ocho,nueve
7010 DATA en Español es, Número
Ok
save "prog1"
Ok
```

Как только пользователь программы вводит число больше 9, управление передается оператору CHAIN (строка 200), в результате выполнения которого загружается и начинает работу другой программный сегмент. Полагая, что первый сегмент все еще находится в памяти, второй сегмент (который будет выдавать названия чисел по-немецки) можно создать с помощью следующих команд:

```
200 chain "prog1"
7000 data nullpunkt, eins, zwei, drei, vier,
     funf, sechs, sieben, acht, neun
7010 data auf Deutsch ist, Nummer
save "prog 2"
Ok
```

Выполнение всей программы начинается с первого сегмента. Если в процессе работы программы ввести (в ответ на запрос INPUT) число больше 9, то программа переключится на другой язык, подсоединяя следующий сегмент.

Пример

```
run "prog1"
Número en Español es:
Número? 4
4 en Español es cuatro
Número? 99
Nummer auf Deutsch:
Nummer? 7
7 auf Deutsch ist sieben
Nummer? 99
Número en Español es:
Número?_
```

Оверлейная структура программы позволяет очень эффективно использовать имеющийся объем динамической памяти.

Общие переменные

В простейшем случае использования команды CHAIN каждый раз перед загрузкой следующего сегмента из динамической памяти удаляются все программные строки и переменные предыдущего сегмента. Пример такого выполнения программы рассматривался выше. Однако может оказаться, что в очередном сегменте требуется использовать значения переменных предыдущего сегмента. В такой ситуации следует использовать оператор COMMON. В операторе COMMON перечисляются переменные, которые должны сохраниться при переходе к следующему сегменту. Сохранить таким образом можно как простые переменные, так и целые массивы, но не отдельные их элементы.

Пример

```
COMMON ITEM$( ),TOTAL,OMIT$,COST( )
```

Как видно в этом примере, имена переменных отделяются друг от друга запятыми, а после каждого имени массива ставятся скобки, внутри которых ничего не записывается.

Хотя операторы COMMON могут стоять в любом месте программы, лучше располагать их в ее начале вместе с операторами DIM и DEF.

◆◆◆ В пределах одного программного сегмента не следует указывать одну и ту же переменную более чем в одном операторе COMMON.

После выполнения оператора CHAIN, т. е. после загрузки нового сегмента, описания типов переменных, заданные в предыдущем сегменте операторами DEFINT, DEFSNG, DEFDBL и DEFSTR, теряют силу, но сохраняются значения общих переменных (указанных в COMMON), попадающих в область действия перечисленных опера-

торов. Поэтому при необходимости следует повторить описания типов в новом сегменте.

Пример. Использование оператора COMMON в случае двух связанных программных сегментов.

```
10 defstr u:defint d
20 common units,dist
30 input "Расстояние";dist
40 input "Миль или километров";units
100 chain "ovly1"
save "mainpgm"
Ok
new
Ok
10 defstr u
20 print dist;units
save "ovly1"
Ok
run "mainpgm"
Расстояние? 1342
Миль или километров? Миль
0 Миль
Ok
```

Переменные DIST и UNITS в первой программе объявлены общими (строка 20), и поэтому их значения, присвоенные в первой программе (строки 30 и 40), должны сохраниться и после загрузки второго сегмента (строка 100). Но, поскольку во второй программе (сегменте) определение целых переменных с помощью оператора DEFINT не повторяется (сравните строки 10 различных сегментов), переменная DIST во втором сегменте воспринимается как переменная с обычной точностью, и, несмотря на оператор COMMON, значение этой переменной теряется.

Сохранить значения всех переменных при переходе от одного сегмента к другому можно и без помощи оператора COMMON, используя лишь возможности оператора CHAIN. В этом случае последний выглядит следующим образом:

```
CHAIN "pgmnam", ,ALL
```

Здесь перед словом ALL стоят две запятые, поскольку помимо ALL у оператора CHAIN есть еще одна опция (необязательный атрибут), которая в данном случае для простоты опущена. Описанный метод сохранения переменных с помощью оператора CHAIN связан с теми же проблемами переопределения типов переменных, что и в случае оператора COMMON.

Ни оператор COMMON, ни опция ALL оператора CHAIN не позволяют сохранить функции, определенные оператором DEF FN

(задаваемые пользователем функции). Это означает, что при необходимости все функции, определяемые пользователем, должны повторно описываться в каждом сегменте, где это требуется. Кроме того, если имя такой функции попадает в область действия какого-либо оператора типа DEFSTR или DEFINT, то соответствующие описания типов должны также повторяться в каждом сегменте.

Совместное выполнение оверлейных сегментов

До сих пор речь шла об использовании оператора CHAIN, когда каждый раз перед загрузкой нового сегмента из памяти полностью удалялся весь предыдущий сегмент. Рассмотрим еще одну дополнительную возможность (опцию) оператора CHAIN, позволяющую присоединять к находящемуся в памяти сегменту строки другого сегмента. В этом случае строки нового сегмента не замещают полностью прежний сегмент, а как бы вставляются в него. Если при этом номер некоторой загружаемой строки нового сегмента совпадает с номером одной из строк сегмента, уже находящегося в памяти, то старая строка полностью заменяется на новую.

Пример. Сегмент программы, в которой совместно используются несколько оверлеев. (В программе подсчитываются затраты на транспортировку грузов. Каждый сегмент вычисляет эти затраты для определенных транспортных средств: 1-й сегмент с именем "truck" (грузовики) для наземного транспорта, 2-й сегмент с именем "mail" (почта) для почтовых отправлений, 3-й сегмент с именем "air" (авиация) для воздушных транспортных средств.)

```
list 1600—1800
1600 '——Подсчет стоимости транспортировки
1610 IF SHIP$="T" THEN CHAIN MERGE
     "truck",,ALL
1620 IF SHIP$="M" THEN CHAIN MERGE
     "mail",,ALL
1630 IF SHIP$="A" THEN CHAIN MERGE "air",,ALL
1700 TOTAL=TOTAL+SHIP
1800 GOSUB 3000 'Печать общей стоимости
Ok
load "mail"
list
0 GOTO 1640 'Переход на начало сегмента
1640 READ RATE(1),RATE(2) 'ввод коэффициентов
1650 '   вычисление платы за вес груза
1660 SHIP=RATE(1)+CINT((LBS—1/16)/16)
     *RATE(2)
7000 DATA .30,.25
Ok
```

В основной программе приведенного примера управление передается одному из трех сегментов, предназначенных для вычисления стоимости перевозки грузов (строки 1610—1630). В каждом случае в операторе CHAIN используется атрибут ALL и тем самым при переходе к очередному сегменту сохраняются все значения переменных. Строки загружаемого сегмента вставляются в предусмотренное место между строками основной программы (незанятые номера строк с 1631 по 1699). Во избежание многократного выполнения одних и тех же операторов основной программы каждый сегмент содержит в строке с номером 0 оператор GOTO, который передает управление сразу же первому оператору непосредственно самого сегмента (строка 1640). С этого оператора выполнение сегмента продолжается обычным образом до тех пор, пока в конце концов не начнут выполняться операторы основной программы (строка 1700).

Чтобы загружать сегменты с помощью оператора CHAIN MERGE, необходимо предварительно выполнить команду SAVE с опцией ",A" (гл. 6), указав в ней нужные сегменты.

Включая в оператор CHAIN MERGE команду DELETE, можно исключить строки, номера которых лежат в заданном после слова DELETE диапазоне. При этом после слова DELETE разрешается указывать только один диапазон изменения номеров удаляемых строк. Оператор CHAIN в случае использования опции DELETE имеет следующий вид:

```
1600 CHAIN MERGE "prgname",,DELETE 100—900
```

Дополнительная запятая перед опцией DELETE соответствует неиспользуемой другой опции оператора CHAIN.

Номера, определяющие границы диапазона удаляемых строк в опции DELETE, должны быть такими, чтобы в программе существовали строки с обоими граничными номерами, в противном случае будет выдано сообщение об ошибке. Если в опции DELETE не задан последний номер диапазона, то удаляется ровно одна строка программы с номером, указанным после слова DELETE. При перенумерации строк программы, содержащей оператор CHAIN с опцией DELETE, автоматически изменяются соответствующим образом и номера, указанные в DELETE в качестве границ диапазона удаляемых строк.

Если в операторе CHAIN наряду с DELETE используется одновременно и опция ALL, то такой оператор имеет следующий вид:

```
1700 CHAIN MERGE "ovly 2",,ALL,DELETE 1000
```

Задание начальной строки

Оператор CHAIN имеет еще одну опцию, о которой уже упоминалось ранее в этой главе в связи с появлением дополнительных запятых в предыдущих примерах оператора CHAIN. Эта опция поз-

воляет задать номер строки, с которой начнется выполнение следующего сегмента. Указанный номер строки записывается непосредственно за именем файла; оператор CHAIN в этом случае выглядит так:

```
1610 IF SHIP$="T" THEN CHAIN MERGE
      " truck",1640,ALL
```

Опции ALL и DELETE в таком операторе CHAIN можно использовать обычным образом. Строка с номером, указанным в операторе CHAIN в качестве номера начальной строки, не обязательно должна присутствовать в исходной программе, но должна содержаться в загружаемом сегменте, иначе при выполнении оператора CHAIN будет выдано сообщение об ошибке "Undefined line number" («Неопределенный номер строки»). Команда RENUM не оказывает никакого влияния на заданный в операторе CHAIN номер начальной строки.

Трассировка программ

Чем длиннее и сложнее программа, тем больше времени приходится тратить на проверку правильности ее работы в различных ситуациях.

Одним из наиболее важных этапов при создании надежных и работоспособных программ является тщательное тестирование. Часто, особенно для больших и сложных программ, очень трудно определить, где именно в программе возникает ошибка. Для локализации ошибок полезно использовать специальные вспомогательные средства, одним из которых является оператор TRON, приводящий в действие трассировочную программу. Последняя в процессе работы основной программы выводит номера ее строк в той последовательности, в которой они реально исполнялись. Цепочка этих номеров составляет след (или трассу) работы программы.

Пример

```
list
10 TRON ' Режим трассировки
20 GOSUB 100
30 END
90 '==Рекурсивная подпрограмма==========
100 IF K>5 THEN K=K+1:GOSUB 100
110 RETURN
Ok
run
[10] [20] [100] [100] [100] [100] [100] [110] [1
10] [110] [110] [110] [110] [110] [30]
Ok
```

Отображенный след программы показывает, что первой выполнялась строка 10, затем — строка 20, в результате выполнения которой была вызвана рекурсивная подпрограмма, начинающаяся строкой 100. После этого подпрограмма еще пять раз обращалась к самой себе, передавая каждый раз управление на строку 100. Затем значение переменной K стало равным 6, и управление было передано на строку 110. Начиная с этого момента управление стало передаваться последовательно по адресам возврата, соответствующим вызовам подпрограммы. Первые пять раз возврат осуществлялся на строку 110, поскольку соответствующие обращения к подпрограмме были рекурсивными вызовами ее самой. При шестом возврате управление оказалось переданным в основную программу на строку 30, где и закончилось ее выполнение.

Для отмены режима трассировки используется оператор TROFF. Команда RUN в ее простейшем виде не отменяет этого режима, так что использование команды TRON в режиме немедленной обработки приведет к выполнению трассировки всех последующих программ до тех пор, пока не встретится оператор TROFF. Любая команда, удаляющая текущую программу из памяти (например, команда NEW или LOAD), отменяет также и режим трассировки.

Глава 10

УПРАВЛЕНИЕ ВЫВОДОМ ДАННЫХ НА ЭКРАН ДИСПЛЕЯ И НА УСТРОЙСТВО ПЕЧАТИ

В данной главе описываются дополнительные возможности оператора PRINT, а также другие операторы, позволяющие управлять выводом данных на экран дисплея, и рассматриваются средства вывода данных на печатающее устройство.

Управление выводом данных на экран дисплея

Для очистки экрана, выравнивания выводимых числовых данных и управления курсором в Бэйсике ПВМ предусмотрен целый ряд средств, в том числе оператор PRINT, используемый в совокупности с различными стандартными функциями, и несколько других независимых операторов.

Очистка экрана

Существуют три способа очистки экрана дисплея, причем в любом случае курсор, после того как экран очистится, занимает исходное положение. В режиме немедленной обработки очистку можно произвести путем одновременного нажатия клавиш **Ctrl|Home**, с помощью команды CLS (которая применима и в программируемом режиме) и путем вывода на экран символа ASCII с кодом 12. Такой символ формируется функцией CHR$, и для очистки экрана достаточно выполнить оператор PRINT CHR$(12).

Установка ширины экрана

Большинство видеомониторов позволяет выводить на экран строки длиной 40 или 80 символов; при этом, однако, следует иметь в виду, что для бытовых телевизоров и некоторых типов мониторов использование 80-символьных строк приводит, как правило, к очень нечеткому изображению (особенно при выводе строчных букв). Если предполагается использовать программу при любой ширине экрана, то она должна быть ориентирована на вывод только 40-символьных строк.

В Бэйсике ПВМ предусмотрены средства, позволяющие в явном виде задавать максимально возможную длину строки при выводе на экран дисплея. Для установки длины строки надо последовательно выполнить два оператора: SCREEN и WIDTH.

Пример

```
1010 SCREEN 0:WIDTH 40
```

При выполнении оператора SCREEN 0 отменяется максимальное значение длины строки экрана, установленное при запуске Бэйсик-интерпретатора, и устанавливается длина строки, указанная в следующем за SCREEN 0 операторе WIDTH. Оператор SCREEN 0 должен предшествовать лишь самому первому выполняемому после запуска интерпретатора оператору WIDTH.

Выравнивание выводимых данных по столбцам

В большинстве случае при работе с большими объемами информации гораздо удобнее представлять ее в табличной форме (или выравнивать по столбцам). В гл. 8 было показано, как с помощью запятых, разделяющих перечисленные в операторе PRINT значения, можно расположить выводимые значения строго по столбцам, каждый их которых начинается с одной из стандартных зон. Ширина стандартных зон фиксирована, и в распоряжении программиста имеются три зоны на 40-символьном экране или шесть зон на 80-символьном экране. Однако это не всегда удобно: часто выводимые значения не помещаются в одной зоне или, напротив, зона для них слишком велика. В подобных ситуациях целесообразнее было бы по своему усмотрению определять зоны нужного размера.

Количество используемых зон и ширину каждой из них можно задать с помощью оператора PRINT и функции TAB. При этом все символьные позиции в строке экрана, называемые *колонками* (столбцами), считаются пронумерованными слева направо от 1 до 40 или до 80 в зависимости от ширины экрана дисплея, а функция TAB перемещает курсор на указанную позицию в строке.

Пример. Выполнение оператора PRINT с включенной в него функцией TAB.

```
list
4000 CLS:SCREEN 0:WIDTH 40
4010 PRINT "Наименование";TAB(20);
       "Получено" ;TAB(30); "Стоимость"
4030 READ NR%,DSCR$,SN$,AQ$,COST#,VALUE#
4040 PRINT DSCR$;TAB(20);AQ$;TAB(30);"$";
     COST#
10001 DATA 1,Письменный стол,9/11/78,875,1500
Ok
run
Наименование      Получено  Стоимость
Письменный стол 9/11/78     $875
Ok
```

Как видно из приведенного примера, в функции TAB указывается номер колонки, куда должен выводиться следующий символ: перемещение курсора осуществляется функцией TAB за счет печати про-

белов, и потому при перемещении курсора на указанную позицию встречающиеся на пути символы стираются.

Если курсор находится правее заданной колонки, то функция TAB переводит его на одну строку ниже, а затем — в указанную позицию.

Пример

```
list
4010 PRINT "Наименование" ;TAB(20);
       "Получено" :TAB(30); "Стоимость"
4030 READ NR%,DSCR$,SN$,AQ$,COST#,VALUE#
4040 PRINT DSCR$;TAB(20);AQ$;TAB(30);"$";
     COST#
10012 DATA 12,Диван с кожаной обивкой,
      12/2/79,1377,1377
Ok
run
Наименование          Получено Стоимость
Диван с кожаной обивкой
                      12/2/79    $137
Ok
```

В качестве номера колонки в функции TAB может быть указано любое значение от —32768 до 32767. Если значение меньше единицы, то считается, что оно соответствует первой колонке; если оно превышает ширину экрана, то выполняется операция вычитания по модулю: заданное значение делится на значение ширины экрана, при этом целая часть частного отбрасывается, а остаток интерпретируется как номер колонки, в которую должен выводиться следующий символ. Например, оператор PRINT TAB(241) для 40-символьного экрана выполняется точно так же, как оператор PRINT TAB(1), поскольку 240 MOD 40 = 1.

Функция TAB используется только совместно с оператором PRINT или с операторами PRINT# и LPRINT. При необходимости сгенерировать несколько стоящих рядом пробелов в виде некоторого строкового значения используют функцию SPACE$.

Управление курсором

Описанные выше средства позволяют перемещать курсор в любое место на экране дисплея: для этого достаточно с помощью команды CLS установить курсор в исходное положение и затем с помощью простого оператора PRINT перемещать его по вертикали, а с помощью функции TAB — еще и по горизонтали. Такой метод, однако, не очень удобен, не говоря уже о том, что при его использовании изображение на экране каждый раз стирается.

В Бэйсике ПВМ имеется оператор LOCATE, предназначенный для перемещения курсора в любое заданное место на экране непо-

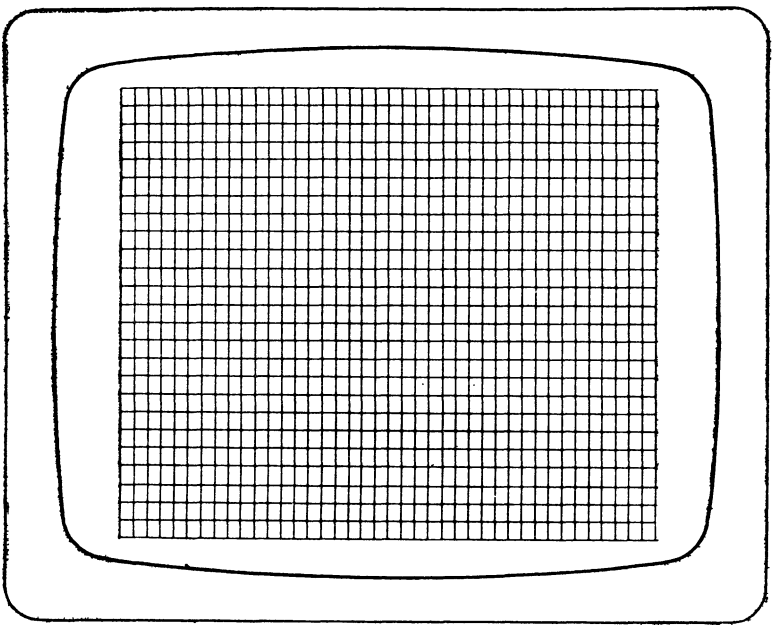


Рис. 10.1. Разбиение экрана на строки и столбцы.

средственно, т. е. без разрушения ранее выведенной на экран информации. При использовании оператора LOCATE экран надо представить разграфленным на строки и столбцы (рис. 10.1). Символ может быть выведен в любую точку пересечения строки со столбцом. В простейшем случае в операторе LOCATE указываются номер строки и номер столбца, в точку пересечения которых должен быть переведен курсор. Такой оператор LOCATE выглядит следующим образом:

```
LOCATE 13,29
```

Первое число, заданное в операторе LOCATE, указывает номер строки, а второе — номер столбца. Номера строк и столбцов в LOCATE можно также задавать с помощью любых числовых выражений. Обычно на экране имеется 24 строки, пронумерованные от 1 до 24 сверху вниз, и 40 или 80 столбцов (в зависимости от ширины экрана), пронумерованных от 1 до 40 или 80 слева направо. Значения номеров, задаваемые в операторе LOCATE, должны заключаться в этих пределах, в противном случае появится сообщение об ошибке "Illegal function call..." («Недопустимый вызов функции...»).

Как и обычно, вывод какого-либо символа в крайней правой позиции строки приводит к автоматическому возврату каретки (т. е.

к переводу курсора в начало следующей строки). Если это происходит после заполнения последней позиции 24-й строки, все изображение экрана сдвигается на одну строку вверх, в результате чего самая верхняя строка исчезает с экрана.

◆◆◆ Поскольку нет способа предотвратить такой автоматический возврат каретки, следует соблюдать осторожность при заполнении последних позиций строки, а лучше вообще избегать вывода символа в крайний правый столбец последней, 24-й строки экрана. Как уже отмечалось (гл. 8), если выводимое значение не помещается в оставшихся позициях строки, то оно целиком выводится на следующей строке.

Пример. Использование двух операторов LOCATE: первого для установки нужного положения курсора при вводе данных с клавиатуры, а второго для выполнения тех же действий при выводе данных с помощью оператора PRINT.

```
list
10 CLS
100 LOCATE 1,20:INPUT "Строка, столбец" R,C
110 LOCATE R,C:PRINT "*";
120 GOTO 100
Ok
run
                              Строка, столбец? 2,20
                              *
```

Использование последней строки экрана дисплея

Обычно при работе с экраном программист имеет в своем распоряжении 24 строки для вывода данных. В действительности на экране имеется еще одна, 25-я строка, но она зарезервирована для высвечивания определений функциональных клавиш (гл. 6) и обычно недоступна для Бэйсик-программ. Такое специальное назначение 25-й строки можно отменить с помощью команды KEY OFF и использовать ее так же, как и остальные, но со следующим ограничением. Вывести данные на 25-ю строку можно, только выполнив предварительно оператор LOCATE. Возврат каретки после 24-й строки всегда приводит к сдвигу всего изображения (точнее, 24 первых строк) на одну строку вверх, и курсор не сдвигается вниз на 25-ю строку. Если возврат каретки произошел по каким-либо причинам после заполнения 25-й строки, то и в этом случае все строки, кроме 25-й, сдвинутся на строку вверх, т. е. результат будет тот же, что и при возврате каретки после заполнения 24-й строки. Всякий раз при описанном сдвиге изображения курсор перемещается на начало новой свободной строки, т. е. на начало 24-й строки.

Определение положения курсора

Получить значение номера столбца, в котором в данный момент находится курсор, можно с помощью функции POS. Результатом выполнения этой функции является число, заключенное в пределах от 1 до 40 (при 40-символьной ширине экрана дисплея) или от 1 до 80 (при 80-символьной ширине экрана дисплея).

Пример

```
PRINT POS(0)
```

Значение, стоящее в скобках после слова POS фиктивно, т. е. при вычислении функции POS оно никак не используется. Однако какое-то значение (все равно какое) в скобках должно быть записано (в данном примере это значение равно 0).

Функция CSRLIN позволяет определить, в какой строке находится в данный момент курсор. Результатом вычисления этой функции является число между 1 и 25, соответствующее номеру строки экрана.

Пример

```
PRINT CSRLIN
```

Функция CSRLIN не совсем обычна, так как при ее задании не используются скобки. В связи с этим она выглядит так же, как переменная, но тем не менее ей нельзя присваивать никаких значений.

Формат выводимых данных

С помощью оператора LOCATE, функции TAB и определенной расстановки запятых в операторе PRINT можно управлять выбором места на экране дисплея, куда выводится то или иное значение. Однако ни одно из перечисленных средств не позволяет влиять на форму представления информации. Так, при выполнении оператора PRINT и строковые, и числовые данные выводятся в соответствии со строго установленными правилами, определяющими для каждого типа данных свой единственно возможный формат вывода. Вообще говоря, можно было бы изменять формат выводимых с помощью оператора PRINT данных, но для этого пришлось бы производить целый ряд сложных манипуляций, комбинируя функции STR$, RIGHT$, LEFT$, MID$ и т. п. Такой способ вывода данных в нужном формате очень неудобен, и в Бэйсике предусмотрен специальный оператор форматированного вывода PRINT USING.

Как и оператор PRINT, оператор PRINT USING содержит список значений, предназначенных для вывода. Значения в этом списке отделяются друг от друга точкой с запятой: использовать в качестве разделителей запятые в этом операторе не разрешается. Если точка с запятой стоит и после самого последнего значения списка, то возврат каретки не производится, хотя в любом другом

случае он происходит автоматически сразу же после вывода всех значений списка.

Первым в списке значений оператора PRINT USING обязательно должно стоять строковое значение. Это первое значение играет особую роль — оно задает вид, в котором будут выводиться все остальные значения, указанные в списке. Каждый символ этого выделенного строкового значения имеет определенную интерпретацию, так что в целом значение определяет некоторый шаблон, устанавливающий, на какие числовые и строковые зоны разбиваются выводимые значения, какова длина и другие характеристики каждой зоны и т. п. При необходимости каждое выводимое значение списка преобразуется в соответствии с заданным шаблоном. В табл. 10.1 дана расшифровка действия каждого символа, который может встречаться в шаблоне.

◆◆◆ Оператор PRINT USING является одним из самых сложных операторов Бэйсика ПВМ, и обычно для полного его освоения требуется определенный опыт.

Задание формата для вывода строковых значений

С помощью оператора PRINT USING можно выводить строковые значения в их непосредственном виде (полностью) либо только заданное число символов строковой переменной, отсчитываемое слева.

Пример

```
list
10 A$ = "Время летит"
20 PRINT  USING "!";A$        '1    символ
30 PRINT  USING "\ \";A$      '2 символа
40 PRINT  USING "\  \";A$     '3 символа
50 PRINT  USING "\   \";A$    '4 символа
60 PRINT  USING "&";A$        'все символы
Ok
run
В
Вр
Вре
Врем
Время летит
Ok
```

Как видно из этого примера, символ !, входящий в шаблон, определяет строковую зону длиной в один символ, а символ & — зону переменной длины для вывода целиком всего строкового значения.

Пара перевернутых косых черточек (\), возможно разделенных некоторым числом пробелов, задает строковую зону фиксированной

Таблица 10.1. **Интерпретация символов, входящих в шаблон оператора PRINT USING**

Символы	Интерпретация
	Числовые форматы [1]
+	Вывод в явном виде знака + или знака —
**	Заполнение оставшихся слева незанятых позиций зоны звездочками
$$	Вывод перед самим значением знака доллара
**$	Заполнение левых позиций звездочками и вывод непосредственно перед самим значением знака доллара
#	Выдача одного разряда выводимого значения
,	Вывод запятых через каждые три разряда
.	Вывод в заданной позиции зоны десятичной точки
—	Вывод пробела, если значение положительно, и знака —, если оно отрицательно
^^^^	Вывод значения в экспоненциальном представлении
	Строковые форматы
!	Вывод одного символа
\ \	Вывод фиксированного количества символов: двух или более в зависимости от числа пробелов, стоящих между косыми чертами
&	Вывод строки переменной длины (вывод целиком всего строкового значения)
	Символьные константы
Любой символ	Любой символ (кроме —), который не входит в число символов, определяющих числовые и строковые зоны, выводится непосредственно так, как он записан
_	Вывод следующего символа в непосредственном виде, даже если он является одним из специальных символов, определяющих зоны

[1]) Расположение символов в шаблоне в самом общем случае:

$$+\left\{\begin{matrix}**\\**\$\\\$\$\end{matrix}\right\}\#,\ \#\cdots\#.\#\cdots\#\left\{\begin{matrix}+\\-\end{matrix}\right\}\wedge\wedge\wedge\wedge$$

длины. Длина этой зоны определяется как общее количество символов шаблона, считая обе косые черты и все заключенные между ними пробелы. Строковое значение при таком шаблоне выводится в зону, начиная с ее крайней левой позиции. Если длина строкового значения слишком велика и оно не помещается в задаваемое шаблоном поле, то «избыточные» правые символы этого значения

отбрасываются. Если длина строкового значения меньше длины зоны, то оставшиеся свободные места справа в зоне заполняются пробелами.

Пример. Шаблон определяет зону длиной в семь символов, а выводимое значение состоит только из четырех символов.

```
print using "\\"; "Стол";:print
      "$" ;875
Стол  $ 875
```

Форматирование числовых значений

Обычно числа, записанные в столбик, легче воспринимаются, если они выровнены относительно десятичной запятой, т. е. если десятичные запятые всех чисел столбца расположены точно друг под другом. Выровненные таким образом числа можно выводить с помощью оператора PRINT USING: для этого в операторе задается числовая зона фиксированной длины и числовые значения при выводе преобразуются в соответствии с зоной. Зная, сколько разрядов занимает числовое значение, можно вывести его на экран с большей точностью, чем при выполнении обычного оператора PRINT.

Числовая зона задается с помощью символов #: каждый такой символ соответствует одному выводимому разряду числового значения. Если выводимое значение отрицательно, то одна позиция в зоне отводится под знак минус. Числовые значения всегда выводятся выровненными по крайней правой позиции зоны.

Пример

```
print using   "####" ;12
  12
print using   "####" ;—123
—123
```

Если количество разрядов числового значения меньше числа позиций в зоне, то слева от первой значащей цифры это значение дополняется пробелами так, чтобы была заполнена вся зона. Если значение содержит слишком много цифр и не помещается в зоне, заданной шаблоном, то оно все равно выводится полностью, а перед самой первой его цифрой выводится знак процента.

Пример.

```
print using "####" ;123456
%123456
```

Вывод десятичной запятой

Для того чтобы вывести числовое значение с десятичной запятой и с дробной частью, надо после одного из символов # в шаблоне поставить десятичную точку. Это позволяет заранее

установить, в каком месте должна стоять десятичная запятая, т. е. сколько цифр будет в выводимом числе до запятой и сколько после. Если числовое значение имеет больше значащих цифр после запятой, чем предусмотрено шаблоном, то перед выводом оно соответствующим образом округляется; если оно имеет меньше цифр после запятой, чем указано в шаблоне, то оно дополняется нулями до заполнения всей зоны.

Пример. Использование десятичных точек в шаблоне.

```
list
10 INPUT "Введите число: ",A
20 PRINT, USING "####.##";A
30 GOTO 10
Ok
run
Введите число:  1234.50
1234.50
Введите число:  12.3456
  12.35
Введите число:  .123456
   0.12
Введите число: _
```

Знаки плюс и минус в шаблонах

Как было показано, положительные числовые значения обычно выводятся без знака, а в случае отрицательных значений одна позиция зоны отводится под знак минус. Во всех рассмотренных примерах никаких знаков в самих шаблонах не использовалось. Однако в общем случае в шаблон можно включать знаки плюс и минус в явном виде: допускается ставить знак "+" перед самым первым символом # шаблона и знак "+" или знак "—" после последнего символа #.

Пример.

```
list
10 A$="+####.#" :B$="####.#—" :
   C$="###.#+"
20 PRINT A$,B$,C$
30 FOR N=—88.8 TO 88.8 STEP 177.6
40 PRINT  USING A$;N;:PRINT ,
50 PRINT  USING B$;N;:PRINT ,
60 PRINT  USING C$;N
70 NEXT
run
```

```
+#####.#      ###.#-      ###.#+
    -88.8       88.8-       88.8-
    +88.8       88.8        88.8+
Ok
```

Знак плюс, используемый в качестве первого или последнего символа шаблона, означает, что всегда выводится знак числа: "+" для положительных значений и "—" для отрицательных. Знак минус используется только в качестве последнего символа шаблона и означает, что после отрицательного числа должен выводиться знак "—", а после положительного пробел.

Запятые в шаблонах

Для удобства большие числа часто записывают с запятыми через каждые три цифры, например 100,000,000,000. С помощью оператора PRINT USING можно выводить числа и в таком виде: для этого достаточно поставить в шаблоне запятую после первого символа #.

Пример

```
print using "#,##########.##" ;1e7
  10,000,000.00
Ok
```

Запятую в шаблоне можно ставить не только после первого, но и после любого другого символа #, но обязательно до десятичной точки. Различные местоположения запятых в шаблоне определяют различные способы расстановки запятых в выводимых значениях. Количество разрядов, занимаемых выводимым числом, включая все встречающиеся запятые, не превышает ширины зоны, задаваемой шаблоном.

Шаблоны экспоненциального представления числовых значений

Для вывода числовых значений в экспоненциальном представлении надо поставить в конце обычного числового шаблона четыре символа ^.

Пример

```
print using "###.##^^^^" ;-.97654
-97.65E-02
Ok
```

В мантиссе числа выводится максимально возможное количество значащих цифр, а идущий следом порядок настраивается соответственно мантиссе. Если шаблон задает более одной цифры до запятой, то крайний левый разряд всегда будет либо пробелом, либо знаком минус в зависимости от знака выводимого значения.

Шаблоны для вывода значений денежных сумм

Для вывода знака доллара перед первой цифрой числового значения достаточно заменить в шаблоне первый символ # на два символа $. В результате этого длина зоны увеличится на единицу для вывода знака доллара.

Пример

```
print using "$$###.##" ;15.95
  $15.95
Ok
```

Когда денежная сумма принимает отрицательное значение, в конце шаблона обычно добавляют знак "+", и после самого значения всегда будет выводиться его знак. Если знак "+" должен стоять в начале шаблона, то его надо записать перед символами $$. Если шаблон денежных величин не содержит в явном виде никакого знака, при выводе отрицательных значений перед знаком доллара будет стоять знак минус.

Если значащие цифры числового значения не полностью заполняют зону, то обычно такое числовое значение при выводе дополняется слева соответствующим количеством пробелов до заполнения всей зоны. Вместо пробелов можно выводить звездочки: для этого достаточно заменить два первых символа # шаблона на символы *.

Пример

```
print using "**######.##" ;249.95
****249.95
Ok
```

Для вывода звездочек перед знаком доллара в качестве первых трех символов шаблона надо записать **$.

Использование констант в шаблонах

Шаблон, задаваемый в операторе PRINT USING, может содержать любые символы. Символы, входящие в шаблон и не принимающие участия в определении числовых или строковых зон, всякий раз при использовании этого шаблона будут выводиться в том виде, как они записаны, без каких-либо изменений. Такими символами могут быть пробелы, буквы алфавита, цифры и большинство знаков препинания.

Пример

```
tot=989457.82
Ok
print using "Общая сумма #,#######.##" ;tot
Общая сумма 989,457.82
Ok
```

Без каких-либо изменений (т. е. в точности так, как записано) можно выводить специальные символы, которые в обычных условиях определяют строковые и числовые зоны. Такими символами являются !,#, \, &,_, ^, $, *, ., + и —. Заметим, что некоторые из них (^, $, *, ., + и —) часто и при обычном использовании появляются в выводимом значении в явном виде. Для того чтобы символ, встречающийся в шаблоне, воспринимался как символьная константа и выводился всегда без изменений, надо поставить перед ним в шаблоне символ _.

Пример

```
print using "Вес:####_#";180
Вес: 180 #
Ok
```

Шаблоны для нескольких значений

С помощью единственного шаблона можно задавать форматы вывода сразу для нескольких различных значений.

Пример

```
list
100 FMT10$="\          \\      \ $$######.##"
4010 PRINT  "Наименование"; TAB(14); "Получено";
     TAB(27); "Стоимость"
4030 READ NR%,DSCR$,SN$,AQ$,COST#,VALUE#
4040 PRINT USING FMT10$;DSCR$;AQ$;COST#
10101 DATA 101,Автоответчик,G395298,
      6/22/81,215,215
Ok
run
Наименование Получено Стоимость
Автоответчик  6/22/81    $215.00
Ok
```

В строке 100 определяется шаблон, который затем используется в строке 4010. Этот шаблон задает три зоны: две строковые и одну числовую.

Если оператор PRINT USING содержит больше выводимых значений, чем число зон, определенных указанным в этом операторе шаблоне, то после вывода значения, соответствующего последней зоне, шаблон начинает использоваться повторно с самого начала.

Пример. С помощью шаблона, определяющего всего две зоны, выводятся шесть различных величин.

```
print using "##.##% & ";
78.11; "N" ;20.95; "0"; .95; "A"
```

```
78.11% N 20.95% 0 0.95% A
Ok
```

Если в шаблоне определено больше зон, чем количество выводимых значений, то лишние зоны просто игнорируются. Числовым значениям должны соответствовать числовые шаблоны, а строковым — строковые, в противном случае произойдет ошибка.

Вывод данных на печать

Чтобы вывести данные на устройство печати, достаточно одновременно нажать клавиши ⇧|**PrtSc** — и все, что выведено в данный момент на экран дисплея, воспроизведется на печатающем устройстве. Эта операция выполняется даже в момент ожидания ПВМ ввода данных с клавиатуры. При этом некоторые символьные коды соответствуют при выводе на экран одним символам, а при выводе на печатающее устройство — другим. Буквы, цифры и большинство знаков препинания выводятся на печать в том же виде, как они были представлены на экране дисплея. Что же касается специальных графических символов, которые могут быть выведены на экран дисплея ПВМ, то практически ни один из них не может быть воспроизведен печатающим устройством. Интерпретация кодов, больших 127 или меньших 32, существенно зависит от типа печатающего устройства. Например, матричное устройство 80CPS фирмы IBM печатает составные графические символы в случае тех кодов, которые на другом печатающем устройстве воспроизводились бы в виде букв, выделенных курсивом.

Использование клавиши **PrtSc** не является единственным способом вывода данных на устройство печати: для этого можно непосредственно или с несущественными изменениями использовать все операторы и функции, которые применялись до сих пор для вывода значений на экран дисплея. В табл. 10.2 сравниваются операторы и функции, предназначенные для работы с дисплеем и с печатающим устройством.

Отметим некоторые различия между выводом информации на экран дисплея и на печать. В программе с помощью оператора LOCATE можно перескакивать с одной строки экрана дисплея на другую, что совершенно исключено в случае печатающего устройства. В каждый момент на печатающее устройство выдается целостная строка. Хотя большинство печатающих устройств и обладает ограниченными возможностями возврата каретки на одну позицию, эти возможности нельзя сравнить с подвижностью курсора дисплея.

Печатающее устройство всегда воспроизводит строку целиком. Оно имеет память емкостью в одну строку для запоминания символов, записанных в операторах LPRINT или LPRINT USING,

Таблица 10.2. **Эквивалентные операторы вывода данных на экран дисплея и на печатающее устройство**

Дисплей	Печатающее устройство
PRINT	LPRINT
PRINT USING	LPRINT USING
POS	LPOS
TAB	TAB
SPC	SPC
CSRLIN	Эквивалент отсутствует
LOCATE	» »

которые оканчиваются точкой с запятой. Как только на печатающее устройство поступает символ возврата каретки, печатаются сразу все накопленные к этому моменту символы. Например, в результате выполнения приводимого ниже оператора такой символ не поступит и ничего не напечатается:

```
LPRINT "Поспешишь — людей насмешишь" ;
```

Ничего не изменится, если добавить еще и оператор

```
LPRINT "(Пословица)" ;
```

При выполнении же какого-либо оператора LPRINT или LPRINT USING без стоящей в конце точки с запятой на устройство печати поступает сигнал возврата каретки, в результате чего печатаются все накопленные в памяти устройства символы. Например, если после двух предыдущих операторов выполнить оператор LPRINT в его простейшем виде (состоящий лишь из командного слова LPRINT), то мгновенно будет напечатана целая фраза:

```
Поспешишь — людей насмешишь (Пословица)
```

Если в памяти печатающего устройства накопилось достаточное количество символов для заполнения всей строки целиком, то на печатающее устройство автоматически посылается сигнал возврата каретки и печатается очередная строка. Все последующие символы, поступающие в устройство печати, накапливаются для формирования следующей строки. Стандартная длина строки печатающего устройства равна 80 символам, так что возникающие при выводе длинных строк эффекты совершенно аналогичны описанным ранее эффектам, возникающим при выводе длинных строк на экран 80-символьного дисплея.

Когда на печатающее устройство пересылается символ возврата каретки, одновременно с ним поступает еще и символ, интерпретируемый как требование протяжки бумаги на одну строку. Без этого символа перевода строки все печатающиеся строки накладывались бы друг на друга.

Разбиение на страницы при печати

Можно так организовать работу программы, что при выводе данных на устройство печати строки будут группироваться в страницы, т. е. в нужный момент будет фиксироваться конец очередной

```
10 GOTO 1000
790 '== Подпрограмма обработки конца страницы ====================
800 LCTR%=LCTR%+1 'Увеличение счетчика числа строк
810 IF LCTR%<=56 THEN RETURN 'Завершение работы, если страница не заполнена
820 LPRINT STRING$(66-LCTR%,CHR$(10)) 'Переход к следующей странице
830 LPRINT TAB((80-LEN(TITLE$))/2);TITLE$ 'Печать заголовка страницы
840 LPRINT TAB(70);" стр. ";PAGE%
850 LPRINT
860 LPRINT  "Печать заголовков столбцов"
870 LCTR%=5 'Установка нач. знач. счетчика строк; учтены строки под заголовки и поля
880 PAGE%=PAGE%+1 'Увеличение текущего номера страницы
890 RETURN
990 '-- Демонстрационная программа ----------------------------
1000 PAGE%=1
1010 TITLE$="title"
1020 GOSUB 830
1030 FOR K%=1 TO 100
1040 LPRINT K%
1050 GOSUB 800
1060 NEXT K%
```

Рис. 10.2. Подпрограмма обработки конца страницы при выводе данных на печатающее устройство.

страницы и производиться протяжка бумаги на начало следующей. При этом можно печатать заголовок страницы, ее номер и заголовки столбцов.

Простейший способ выполнить все это состоит в использовании некоторой подпрограммы (рис. 10.2). В подпрограмме, приведенной на рис. 10.2, для подсчета количества строк, уже напечатанных к данному моменту на текущей странице, используется переменная LCTR%, а для хранения номера текущей страницы — переменная PAGE%. Каждый раз при выводе строки в подпрограмме значение счетчика строк (т. е. переменной CTP%) увеличивается на единицу, а затем оно сравнивается с максимально допустимым числом строк на странице (строки 800 и 810 подпрограммы). Если значение счетчика строк меньше максимально допустимого, то никаких действий больше не производится и подпрограмма заканчи-

вает работу. Как только значение счетчика достигнет размера страницы, в подпрограмме выполняется все необходимое для завершения очередной страницы и перехода на следующую (строки 820—890).

В основной программе перед первым обращением к подпрограмме, т. е. перед первым оператором вывода данных, необходимо задать начальный номер страницы (обычно его задают равным 1). При выводе первой страницы можно обращаться не к первой, а к некоторой промежуточной строке подпрограммы, поскольку в первый раз не требуется протяжка бумаги, и можно сразу печатать заголовок страницы, заголовки столбцов и т. д.

Каждый раз после печати очередной строки в основной программе должно выполняться обращение к подпрограмме, чтобы увеличить значение счетчика строк и проверить, не достигло ли оно максимального. Операторы увеличения и проверки значения счетчика можно было включить прямо в основную программу, чтобы реже обращаться к подпрограмме, однако при таком способе страницы могут иметь неодинаковые нижние поля.

Управляющие символы

Матричное печатающее устройство IBM 80CPS, как и большинство современных печатающих устройств, может производить целый ряд специальных операций. Управление этими операциями осуществляется путем засылки на печатающее устройство одного или двух специальных символов. Сами управляющие символы при выводе не печатаются, а только определяют, в каком виде в дальнейшем будут представляться остальные данные.

Односимвольные управляющие коды предназначены для управления движением бумаги, т. е. с их помощью можно управлять местоположением последующих печатаемых данных. В табл. 10.3 перечислены некоторые наиболее распространенные односимвольные коды, используемые почти в любом печатающем устройстве. Например, при выполнении следующего оператора напечатается текущая строка, после чего бумага продвинется на четыре строки:

```
LPRINT STRING$ (4,CHR$(10))
```

Большинство печатающих устройств обладает различными расширенными возможностями, для использования которых применяются двухсимвольные управляющие коды (один символ интерпретируется как начало выполнения некоторого действия, а другой — как завершение). Интерпретация этих управляющих символов существенно зависит от типа печатающего устройства, и обычно для разных устройств управляющие символы интерпретируются поразному. Например, в печатающем устройстве IBM 80CPS двухсимвольные управляющие коды используются для изменения типа

Таблица 10.3. **Наиболее распространенные управляющие символы для печатающего устройства**

Выполняемая операция	Протяжка бумаги	Печать начатой строки	Управляющий символ
Возврат каретки	К следующей строке	Продолжается	CHR$ (13)
Перевод строки	На одну строку	»	CHR$ (10)
Возврат (на одну позицию)	Не происходит	» [1]	CHR$ (8)
Прогон страницы	К следующей странице	»	CHR$ (12)
Звуковой сигнал	Не происходит	Не продолжается	CHR$ (7)

[1]) По управляющему символу CHR$ (8) печатается текущая строка, а затем происходит возврат к последнему напечатанному символу.

шрифта. В табл. 10.4 приведено описание этих кодов, а рис. 10.3 иллюстрирует их использование. В случае какого-либо другого печатающего устройства эти же самые управляющие символы могут вообще не применяться либо использоваться совершенно иначе.

Коммутируемый вывод данных

Используя в программе операторы PRINT и LPRINT, можно выводить данные то на экран дисплея, то на печатающее устройство. Однако может оказаться, что одну и ту же информацию необходимо выводить либо сразу на оба устройства, либо на устройство, тип которого определяется в процессе выполнения программы в зависимости от сложившейся ситуации. Было бы очень неудобно в таких

Таблица 10.4. **Управляющие символы для выбора шрифта (матричное печатающее устройство IBM 80CPS)**

Тип шрифта	Управляющие символы		Количество символов в одной строке [1])
	первый символ (устанавливающий шрифт данного типа)	второй символ (отменяющий шрифт данного типа)	
Широкий	CHR$ (14)	CHR$ (148) [2])	40
Узкий	CHR$ (15)	CHR$ (146) [3])	132
Узкоширокий	CHR$ (15)+ CHR$ (14)	CHR$ (148)+ CHR$ (146) [2], [3])	66
Выделительный	CHR$ (27)+«E»	CHR$ (27)+«F»	Не устанавливается
Двойного удара	CHR$ (27)+«G»	CHR$ (27)+«H»	»

[1]) При стандартном типе шрифта в одной строке печатается 80 символов.
[2]) Широкий и узкоширокий шрифты автоматически дезактивируются после печати строки: таким образом, дезактивацию можно произвести и с помощью следующих управляющих символов: CHR$ (13), CHR$ (10) и CHR$ (12).
[3]) Выделительный шрифт автоматически исключает узкий.

```
10 CC1$=CHR$(27) 'Первый управляющий символ
20 '-- Формирование управляющих символьных строк----------
30 W.ON$=CHR$(14):W.OFF$=CHR$(20) 'Широкий шрифт
40 C.ON$=CHR$(15):C.OFF$=CHR$(18) 'Узкий шрифт
50 D.ON$=CC1$+"G":D.OFF$=CC1$+"H" 'Шрифт двойного удара
60 E.ON$=CC1$+"E":E.OFF$=CC1$+"F" 'Выделительный шрифт
90 '-- Печать примеров ----------------------------------
100 S$="ABCDEFabcdef0123#?!()"
110 LPRINT C.ON$;S$+" Узкий шрифт"; C.OFF$
120 LPRINT S$+" Обычный стандартный шрифт"
130 LPRINT C.ON$;W.ON$;S$+" Узкий шрифт двойной ширины ";C.OFF$
140 LPRINT W.ON$;S$+" Шрифт двойной ширины"
150 LPRINT
160 LPRINT S$+" Обычный нормальный шрифт"
170 LPRINT D.ON$;S$+"Шрифт двойного удара "; D.OFF$
180 LPRINT E.ON$;S$+"Выделительный шрифт"
190 LPRINT D.ON$;S$+"Выделительный шрифт двойного удара";D.OFF$;E.OFF$
```

А *Программа печати образцов шрифта для 12 типов шрифтов*

```
ABCDEFabcdef0123#?!() COMPRESSED TYPE
ABCDEFabcdef0123#?!() NORMAL TYPE
ABCDEFabcdef0123#?!() DOUBLE WIDTH COMPRESSED TYPE
ABCDEFabcdef0123#?!() DOUBLE WIDTH TYPE

ABCDEFabcdef0123#?!() NORMAL TYPE
ABCDEFabcdef0123#?!() DOUBLE STRIKE TYPE
ABCDEFabcdef0123#?!() EMPHASIZED TYPE
ABCDEFabcdef0123#?!() DOUBLE STRIKE EMPHASIZED TYPE
```

В *Результаты работы программы*

Рис. 10.3. Типы шрифта матричного печатающего устройства IBM 80CPS.

случаях каждый раз дублировать все операторы вывода, записывая их один раз с командным словом PRINT, а другой раз с командным словом LPRINT. Этого можно избежать с помощью оператора PRINT, который позволяет определять тип устройства вывода непосредственно в процессе выполнения программы.

Устройства вывода задаются в операторе PRINT с помощью номеров. Для кассетного Бэйсика номер устройства должен заключаться в пределах от 1 до 4, а для остальных версий Бэйсика используются номера 1, 2 и 3 (гл. 12).

В программе прежде всего необходимо связать определенное физическое устройство с одним из указанных номеров. Для этого используется оператор OPEN.

Пример

```
1100 OPEN "LPT1:" FOR OUTPUT AS #1
```

В операторе OPEN после командного слова OPEN записывается стандартное имя устройства. В приведенном выше примере используется имя системного печатающего устройства (LPT1:). Имена всех устройств перечислены в табл. 10.5.

Таблица 10.5. **Стандартные имена устройств ввода-вывода, используемые в Бэйсике**

Имя	Название устройства	Ввод-вывод	В каких версиях Бэйсика используется данное имя
KYBD:	Клавиатура	Ввод	Во всех
SCRN:	Экран	Вывод	» »
LPT1:	1-е печатающее устройство	»	» »
LPT2:	2-е печатающее устройство	»	В дисковом и в расширенном Бэйсике
LPT3:	3-е печатающее устройство	»	То же
COM1:	1-е последовательное устройство	Ввод-вывод	Во всех
COM2:	2-е последовательное устройство	» »	В дисковом и в расширенном Бэйсике
CAS1:	Устройство для записи на магнитную ленту	» »	Во всех
A:	1-й дисковод	» »	В дисковом и в расширенном Бэйсике
B: [1])	2-й дисковод	» »	То же

[1]) Если в системе больше двух дисководов, то кроме указанных используются имена C:, D; и т. д.

Ниже приводится текст программного сегмента, в котором выбор устройства вывода (дисплея или печатающего устройства) осуществляется пользователем в процессе выполнения программы:

```
1100 OPEN "LPT1:" FOR OUTPUT AS #1
1110 OPEN "SCRN:" FOR OUTPUT AS #2
1120 INPUT "Устройство печати или экран дисплея? (P/S) ",D$
1130 IF D$<>"S" AND D$<>"P" THEN 1120
1140 IF D$="P" THEN D%=1 ELSE D%=2
1430 PRINT#D%, TAB(33);"Список изделий"
1440 PRINT#D%,
1450 GOSUB 5500 'Ввод значений параметров очередного изделия
1460 TMPL3$= "\\\"+SPACE$(18)+
     "\$$#######.##"
1470 PRINT#D%, USING, TMPL3$;ITM$;DESCR$;COST#
```

В приведенном программном сегменте один номер устройства присваивается печатающему устройству, а другой — дисплею (строки 1100 и 1110). Переменной D% присваивается то значение номера устройства, которое укажет пользователь в процессе работы программы (строка 1140). Для вывода данных используются операторы PRINT#D% и PRINT#D%, USING (строки 1430, 1440 и 1470), обеспечивающие возможность вывода на любое из двух указанных устройств. Заметим, что оператор PRINT# даже в его простейшем виде, т. е. не содержащий никаких выводимых значений, должен оканчиваться запятой, стоящей после номера усройства (строка

1440); такая запятая в отличие от аналогичной запятой в обычном операторе PRINT не отменяет возврата каретки и не обеспечивает переход к следующей зоне вывода.

Если в программе нужно изменить номер некоторого устройства вывода, т. е. присвоить ему новый номер с помощью нового оператора OPEN, то предварительно необходимо уничтожить текущий номер этого устройства, выполнив для этого оператор CLOSE.

Пример

```
1100 OPEN "SCRN:" FOR OUTPUT AS #1
1700 CLOSE #1
1710 OPEN "LPT1:" FOR OUTPUT AS #1
```

Длина строки при выводе данных на печатающее устройство

Обычно при программировании на Бэйсике предполагается, что строки печатающего устройства имеют длину 80 символов. Если в программе делается попытка напечатать в одной и той же строке больше 80 символов, то Бэйсик-интерпретатор в нужный момент автоматически генерирует символ возврата каретки, посылает его на печатающее устройство, и в результате печатается строка из 80 символов. Некоторые печатающие устройства могут печатать не только 80-символьные строки. Так, матричное печатающее устройство IBM 80CPS может печатать 132-символьные строки обычным плотным шрифтом; печатающие устройства с широкими каретками также могут выдавать строки длиной более чем 80 символов стандартного размера.

С помощью несколько видоизмененного оператора WIDTH можно изменить принятую по умолчанию длину строки при печати данных и установить ее равной любому числу от 1 до 255.

Пример

```
1120 WIDTH #1, 132
```

Первое значение, указанное в операторе WIDTH, интерпретируется как номер устройства вывода, а второе как новое значение длины строки. Для того чтобы команда установки новой длины строки воспринималась печатающим устройством, необходимо предварительно с помощью оператора OPEN присвоить печатающему устройству указанный в WIDTH номер. Новая длина строки будет учитываться при выполнении только операторов PRINT# и PRINT# USING (в которых указан соответствующий номер устройства); при выполнении операторов LPRINT и LPRINT USING всегда используется 80-символьная строка.

Существует еще одна модификация оператора WIDTH, с помощью которой можно лишь задать новое значение длины строки, а фактически это новое значение установится только при выполнении последующего оператора OPEN, содержащего имя соответствующего устройства. Такой оператор WIDTH имеет вид

```
1000 WIDTH "LPT1:", 132
```

Первый выполняемый после оператора WIDTH оператор OPEN будет не только присваивать нужный номер печатающему устройству, но и устанавливать для него новую длину строки.

Пример

```
1010 OPEN "LPT1:" FOR OUTPUT AS #2
```

Поскольку операторы LPRINT, LPRINT USING, LLIST и LIST, "LPT1:" содержат в неявном виде оператор OPEN, при их выполнении будет также устанавливаться новое значение длины строки для печатающего устройства.

Подчеркивание и забивка строк

В дисковом Бэйсике и расширенном Бэйсике предусмотрены средства, позволяющие отменять автоматический переход к следующей строке, который обычно происходит сразу же после возврата каретки. Благодаря этому при необходимости можно повторно печатать нужные символы в только что выведенной строке и тем самым добиваться подчеркивания или забивки. Для такой отмены автоматического перевода строки необходимо выполнить оператор OPEN специального вида и оператор WIDTH, устанавливающий для печатающего устройства длину строки, равную 255. Оператор OPEN такого вида не содержит конструкции FOR OUTPUT. Приведенная ниже программа показывает, как с помощью описанных средств можно производить подчеркивание и забивку строк:

```
2309 '— —Стандартный автоматич. перевод строки
2310 OPEN "LPT1:" FOR OUTPUT AS #1
2319 '— — Отмена автоматич. перевода строки
2320 OPEN "LPT1:" AS #3:WIDTH #3, 255
2330 PRINT #3, "Человек-невидимка"
2340 PRINT #3, "———————————"
2350 PRINT #1,
2360 PRINT #3, "Забастовка"
2370 PRINT #3, STRING$(10, "—")
2380 PRINT #1
```

В результате выполнения этой программы напечатается следующее:

```
Человек-невидимка
Забастовка
```

В приведенной программе номер устройства, равный #1, используется для печати обычных строк с автоматическим переводом строки (строка 2310 программы), а номер устройства, равный 2,— для печати с отменой перевода строки (строка 2320).

Глава 11

ВВОД ДАННЫХ С КЛАВИАТУРЫ

В предыдущих главах описывалось, как можно вводить данные с клавиатуры с помощью оператора INPUT.

Кроме оператора INPUT в Бэйсике имеется целый ряд других операторов и функций, обеспечивающих широкие возможности управления вводом данных с клавиатуры. С помощью этих средств можно изменять форму курсора, делать его то видимым, то невидимым на экране, а также придавать различные интерпретации функциональным клавишам и использовать управляющие клавиши.

Форма курсора и его изображение на экране

В процессе работы программы курсор обычно не виден, но всякий раз при выполнении оператора INPUT он появляется на экране. Кроме оператора INPUT существуют и другие средства ввода данных с клавиатуры, которые, однако, не предусматривают автоматического появления курсора. При использовании этих средств, а также и в любых других ситуациях изображение курсора на экране можно высветить в любой момент выполнения программы с помощью оператора LOCATE специального вида:

```
300 LOCATE ,,1
```

Если после двух запятых в операторе LOCATE стоит 1, то на экране появится изображение курсора, если 0, то оно исчезнет. Две запятые здесь играют ту же роль, что и в операторе LOCATE, описанном в гл. 10, т. е. участвуют в спецификации позиции курсора. В данном случае перед запятыми не стоит никаких номеров строки и столбца. Это означает, что при выполнении оператора LOCATE текущая позиция курсора на экране не изменится.

Оператор LOCATE обладает еще одной дополнительной возможностью: с его помощью можно изменять форму курсора от тонкой горизонтальной черточки до вытянутого вверх прямоугольника. Форма курсора определяется тем, какая часть выделенной ему максимальной площади действительно заполнена: когда заполнена вся площадь, курсор имеет вид прямоугольника. Этот максимальный прямоугольник немного вытянут вверх и имеет высоту чуть больше прописной буквы, так что если и буква, и прямоугольник находятся на одной и той же строке экрана дисплея, то верхние их границы расположены на одном уровне, а нижняя граница прямоугольника — немного ниже уровня нижней границы буквы (рис. 11.1).

Весь максимальный прямоугольник, отведенный под курсор, разбивается на ряд тонких горизонтальных полосок, и в зависимости от того, какие из них заполняются, курсор приобретает ту или иную форму. Для дисплея, связанного с монохроматическим адаптером, таких полосок 14, и они пронумерованы от 0 до 13 сверху

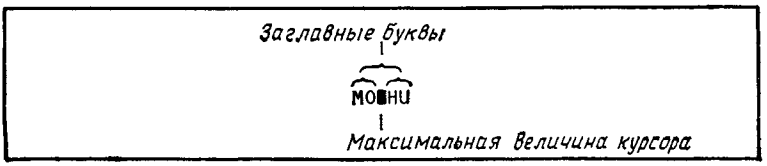


Рис. 11.1. Максимальные размеры курсора.

вниз (рис. 11.2. А). Для дисплея, связанного с адаптером цветных графических устройств, таких полосок 8, и они пронумерованы от 0 до 7 (рис. 11.2. В). Чтобы в качестве курсора на экране высвечивалась часть максимального прямоугольника, ограниченная указанными полосками, в операторе LOCATE надо задать номер верхней и нижней полосок.

Пример. Оператор LOCATE.

```
300 LOCATE ,,1,12,13
```

В приведенном выше операторе первое число указывает на то, что курсор должен высветиться на экране, второе означает но-

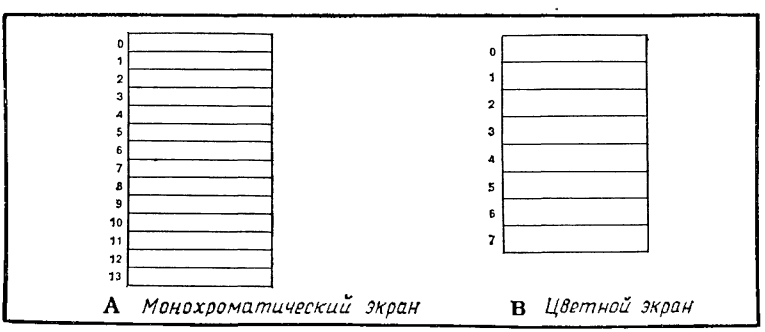


Рис. 11.2. Разбиение отведенного под курсор прямоугольника на горизонтальные полосы.

мер первой используемой полоски, а третье — номер последней используемой полоски. Первые две стоящие друг за другом запятые говорят о том, что текущая позиция курсора при выполнении оператора LOCATE не изменится. Указанный оператор LOCATE определяет стандартную форму курсора, при которой курсор представ-

ляет собой черточку, находящуюся немного ниже строки (подчеркивающий курсор).

Курсор может принимать различные формы, в том числе может состоять из двух отдельных частей (рис. 11.3). Такая кусочная форма курсора получается при выполнении оператора LOCATE, в котором номер полоски, указанный первым, больше номера полоски, указанного вторым. Второй номер в операторе LOCATE

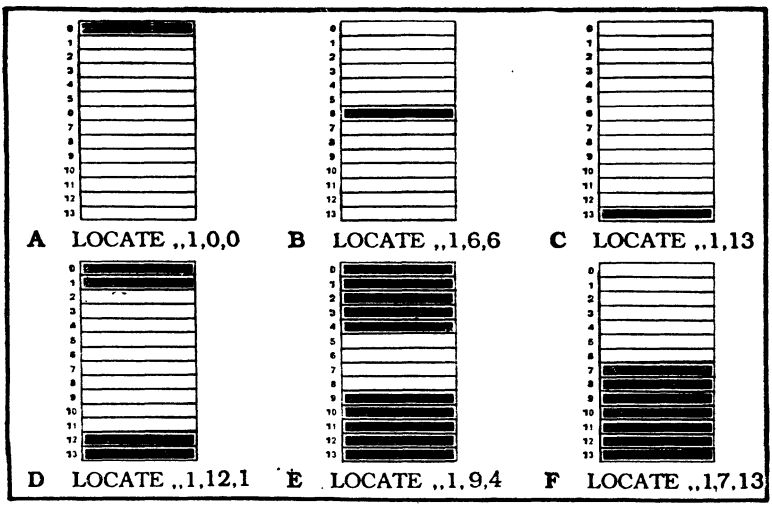


Рис. 11.3. Различные формы курсора для монохроматического экрана (для цветного — аналогично).

является необязательным; если он отсутствует, то при выполнении оператора LOCATE считается, что второй номер совпадает с первым и в результате заполнится лишь одна полоска.

Функциональные клавиши

При нажатии клавиши **F1** на экране появится слово LIST, как если бы оно набиралось на клавиатуре по буквам. Аналогичным образом действует любая из остальных десяти функциональных клавиш, генерируя на экране соответствующее слово. Подобные клавиши на машинах фирмы IBM называются *программируемыми*, поскольку они не связаны жестко с каким-либо определенным символом и действуют не столь быстро, как остальные 76 клавиш клавиатуры; генерируемые ими слова называются *определениями* программируемых клавиш. Бэйсик располагает средствами, позволяющими изменять эти определения так, что при нажатии соответствующей клавиши генерируется заранее заданная цепочка символов длиной до 15 знаков.

На 25-й строке экрана обычно высвечиваются первые шесть символов каждого определения программируемых клавиш. Эти определения с помощью оператора KEY OFF можно удалить с экрана [1], но при этом сами определения не отменяются. Так, если после выполнения оператора KEY OFF нажать какую-нибудь функциональную клавишу, то она сгенерирует на экране то же слово, что и раньше, а в результате выполнения оператора KEY ON в 25-й строке экрана вновь появятся определения программируемых клавиш. Для того чтобы вывести на экран каждое определение полностью, т. е. все его 15 символов, следует выполнить оператор KEY LIST.

Существует еще один вид оператора KEY, с помощью которого можно изменить определение программируемой клавиши. В приводимом ниже примере это делается применительно к клавише **F7**, при нажатии которой должна генерироваться строка "EDIT":

```
KEY 7,"EDIT"
```

Если после выполнения указанной выше команды нажать на **F7** в режиме немедленной обработки, то на экране появится командное слово EDIT. После этого можно набрать номер строки, нажать клавишу ↵, и тогда команда EDIT выполнится.

Для задания в операторе KEY нового определения программируемой клавиши разрешается использовать строковые константы, переменные и выражения. Например, для включения в новое определение еще и символа возврата каретки можно использовать функцию CHR$ так, как это делается в следующем операторе:

```
KEY 8,"FILES" + CHR$(13)
```

При этом исключается необходимость нажимать отдельно клавишу ↵. Так, если после выполнения приведенного выше оператора нажать клавишу **F8**, то одновременно с появлением на экране слова FILES начнет выполняться команда FILES.

При вводе данных с клавиатуры, например, в режиме немедленной обработки или при ответе на запрос оператора INPUT функциональные клавиши обычно действуют как программируемые. Однако если функциональной клавише поставить в соответствие нулевое определение, то последующее нажатие этой клавиши не будет вызывать никаких действий системы.

Использование функциональных клавиш при программировании на расширенном Бэйсике

Расширенный Бэйсик включает в себя средства, позволяющие использовать функциональные клавиши не только для ввода некоторого слова, но и для прерывания выполнения программы и

[1] Точнее, отменить специальный режим использования 25-й строки, чтобы работать с ней так же, как с остальными строками экрана.— *Прим. перев.*

вызова определенной стандартной подпрограммы. Чтобы нажатие функциональной клавиши приводило к реализации указанных действий, необходимо предварительно выполнить два специальных оператора расширенного Бэйсика.

Первый из них — это оператор KEY специального вида, активизирующий функциональную клавишу так, что любое ее нажатие приведет к вызову подпрограммы.

Пример. Оператор KEY для клавиши **F1**:

```
30 KEY (1) ON
```

Записанное в скобках значение определяет, какая именно функциональная клавиша будет активизирована. После выполнения приведенного выше оператора нажатие клавиши **F1** не будет приводить к генерации какой-либо строки символов.

Для того чтобы при нажатии функциональной клавиши происходил вызов некоторой подпрограммы, необходимо после активизации этой клавиши с помощью указанного выше оператора KEY выполнить еще один оператор. Этот второй оператор устанавливает номер строки подпрограммы, к которой будет происходить обращение.

Пример. Оператор для клавиши **F1**:

```
35 ON KEY (1) GOSUB 20100
```

Значение в скобках указывает, для какой функциональной клавиши выполняется данный оператор. Активизированная функциональная клавиша будет действовать как вызывающая подпрограмму только после выполнения оператора, подобного приведенному выше. Если в таком операторе в качестве номера строки подпрограммы стоит 0, то выполнение этого оператора блокирует действие соответствующей функциональной клавиши.

Перед выполнением каждой новой строки программы интерпретатор расширенного Бэйсика проверяет, не была ли нажата какая-нибудь активизированная функциональная клавиша: если такое нажатие производилось, то интерпретатор вызывает соответствующую подпрограмму.

Для блокирования такого вызова при одновременном разрешении действия исходного определения программируемой клавиши необходимо выполнить оператор, подобный следующему:

```
20170 KEY (1) OFF
```

Как и в предыдущих случаях, значение в скобках определяет функциональную клавишу, для которой выполняется данный оператор.

Описанные выше операторы KEY-OFF и KEY-ON не оказывают никакого воздействия на 25-ю строку экрана дисплея. Высвечиваемые в этой строке определения программируемых клавиш исчезают

с экрана или появляются вновь при выполнении только таких операторов KEY OFF и KEY ON, которые имеют самый простой вид, т. е. состоят всего лишь из одного командного слова.

Иногда возникает необходимость временно оградить программу от прерываний, вызываемых функциональными клавишами, и в то же время запомнить каждое нажатие такой клавиши с тем, чтобы впоследствии можно было выполнить все соответствующие вызовы подпрограмм. Все это достигается с помощью оператора, подобного приведенному ниже, который временно дезактивизирует клавишу **F1**:

```
1430 KEY (1) STOP
```

Если после выполнения указанного выше оператора производились нажатия клавиши **F1**, то все соответствующие вызовы подпрограмм все-таки будут реализованы, но не сразу, а только после выполнения оператора KEY (1) ON, вновь активизирующего эту клавишу. Если же во время работы программы нажималось несколько различных функциональных клавиш, временно дезактивизированных операторами KEY-STOP, то соответствующие вызовы подпрограмм будут впоследствии производиться в том порядке, в котором будут вновь активизироваться функциональные клавиши.

Для вызова подпрограмм, кроме функциональных клавиш, можно также использовать четыре клавиши малой клавиатуры, управляющие движением курсора. Для этого надо выполнить все те же операторы, что и в случае функциональных клавиш, как описано выше. При этом каждая из четырех управляющих клавиш задается своим номером: клавише ↑ соответствует номер 11, клавише ← номер 12, клавише → номер 13, клавише ↓ номер 14. Эти номера используются в операторах KEY-ON, KEY-OFF, KEY-STOP и ONKEY-GOSUB точно так же, как номера функциональных клавиш.

Пример. Программа, в которой вместо функциональных клавиш для организации вызовов подпрограмм используются клавиши управления курсором; каждая подпрограмма предназначена для перемещения курсора в том или ином направлении.

```
list
10 FOR K%=11 TO 14
20 KEY (K) ON 'Активизация клавиш управления курсором
30 NEXT
40 ON KEY (11) GOSUB 1010 'вверх
50 ON KEY (12) GOSUB 1020 'влево
60 ON KEY (13) GOSUB 1030 'вправо
70 ON KEY (14) GOSUB 1040 'вниз
80 CLS:LOCATE ,,1 'Появление курсора на экране
100 GOTO 100 'Ожидание нажатия клавиши
      управления курсором
```

```
1010 PRINT CHR$(30);:RETURN 'вверх
1020 PRINT CHR$(29);:RETURN 'влево
1030 PRINT CHR$(28);:RETURN 'вправо
1040 PRINT CHR$(31);:RETURN 'вниз
Ok
```

Управление вводом данных с клавиатуры

Экран дисплея очень чувствителен к различным воздействиям на клавиатуру. Поэтому использование редактирующих клавиш, так необходимых при работе с дисплеем, часто приводит к целому ряду неприятностей. Нажимая клавиши ↵, Esc, а также клавиши управления курсором на малой клавиатуре, можно легко испортить все изображение на экране дисплея, и если это случается, то пользователь, как правило, объясняет это ошибками в программе, а не собственной небрежностью при работе с клавиатурой. В связи с этим в Бэйсик ПВМ включены средства, позволяющие учитывать в самой программе возможность нежелательных воздействий на экран дисплея со стороны клавиатуры и принимать соответствующие меры предосторожности.

Отмена возврата каретки при выполнении оператора INPUT

В дисковом и в расширенном Бейсике в оператор INPUT можно включить дополнительную точку с запятой:

```
790 INPUT; "Порядковый номер" ;SN$
```

При вводе с клавиатуры ответа на такой запрос оператора INPUT пользователь, завершив запись, должен нажать клавишу ↵. При выполнении обычного оператора INPUT такое действие вызывает автоматический возврат каретки, а при выполнении оператора INPUT, содержащего дополнительную точку с запятой, автоматического возврата каретки не происходит. Это особенно важно при заполнении 24-й или 25-й строки экрана, поскольку в данном случае возврат каретки привел бы к сдвигу всего изображения на одну строку вверх с потерей самой верхней строки.

Оператор LINE INPUT

При выполнении оператора LINE INPUT все символы, вводимые с клавиатуры до первого возврата каретки, приписываются одной и той же строковой переменной. Поскольку вводится только одно значение, запятые в качестве разделителей не используются, т. е. при наборе вводимых данных на клавиатуре запятые можно ставить где угодно без заключения всей записи в кавычки. В отличие от обычного оператора INPUT, при выполнении оператора

LINE INPUT вопросительный знак, как требование к пользователю ввести значение, на экран не выводится, но при необходимости его всегда можно включить в явном виде в текст (необязательного) наводящего сообщения оператора LINE INPUT.

Пример. Оператор LINE INPUT при выполнении которого выводится наводящее сообщение, но без знака вопроса.

```
790 LINE INPUT "Наименование изделия: " ;ITEM$
```

Все клавиши редактирования, действующие при выполнении обычного оператора INPUT, точно так же действуют и при выполнении оператора LINE INPUT. Единственное преимущество последнего оператора с точки зрения пользователя состоит в возможности свободно использовать запятые при вводе значения.

Функция INPUT$

Более мощным средством управления вводом с клавиатуры является функция INPUT$, предназначенная для ввода строковых значений заданной длины.

Пример

```
310 K$ = INPUT$(1)
```

Значение, стоящее в скобках, указывает, сколько символов должно быть введено. При выполнении оператора, подобного приведенному выше, дальнейшая работа программы не начинается до тех пор, пока пользователь не введет столько символов, сколько требуется (сколько указано в функции INPUT$). До этого момента клавиша ↵ не будет действовать как завершающая ввод, но одновременное нажатие клавиш в комбинации **Ctrl|Scroll Lock** по-прежнему будет вызывать прерывание выполнения программы.

При выполнении функции INPUT$ курсор на экране не появляется; не появляются также и символы, соответствующие нажимаемым клавишам; эти символы будут просто вводиться в машину. Если нужно, чтобы вводимая с клавиатуры запись одновременно появлялась и на экране, то этого можно добиться с помощью оператора PRINT. Однако перед выводом на экран следует проверить, нет ли среди символов вводимой строки «опасных», т. е. таких, которые могут вызвать нежелательные воздействия на изображение. Если такие символы обнаружатся, то их следует исключить из числа выводимых на экран.

Пример. Использование функции INPUT$.

```
list
10 LOCATE ,,1 'Появление на экране курсора
20 K$ = INPUT$(1) 'Ожидание нажатия клавиши
40 IF ASC(K$)>31 THEN PRINT K$; 'Вывод на экран введенного
   символа
```

```
50 GOTO 20 'Переход к вводу следующего символа
Ok
```

Все опасные в указанном смысле символы имеют числовые коды, меньшие 32. В приведенной выше программе проверяется значение кода каждого введенного с клавиатуры символа; если это значение меньше 32, то соответствующий символ игнорируется, т. е. не выводится на экран дисплея (строка 40).

Как отмечалось ранее, некоторые из кодов символов, заключенных между 0 и 32, а также код, равный 127, интерпретируются при выводе символов на экран дисплея с помощью операторов PRINT или PRINT USING не так, как в случае, когда символы выводятся на экран автоматически сразу же после нажатия соответствующих клавиш. В частности, при выводе на экран символов с помощью операторов PRINT или PRINT USING «опасными» являются не все символы с кодами, меньшими 32, а только те, коды которых принимают значения от 8 до 13 или от 28 до 31 либо равны 0.

Пример. Видоизмененная предыдущая программа, в которой из числа выводимых на экран исключаются только указанные «реально опасные» символы.

```
list
10 LOCATE ,,1 'Появление курсора на экране
20 K$=INPUT$(1):K%=ASC(K$) 'Ввод с клавиатуры 1 символа
40 IF K%>31 OR (K%<28 AND K%>13) OR (K%<8 AND
   K%>0) THEN PRINT K$; 'Вывод на экран введенного символа
50 GOTO 20 'Переход к вводу следующего символа
Ok
```

Функция INKEY$

С помощью функции INKEY$ можно во время выполнения программы узнать, какая клавиша была только что нажата. Эта функция принимает строковое значение, которое может быть нулевой строкой либо состоять из одного или двух символов. Если значение функции — нулевая строка, то это означает, что никакие клавиши не нажимались; если длина строки равна 1, то само значение строковой переменной — это символ, только что набранный на клавиатуре; если значение состоит из двух символов, это говорит о том, что в результате последнего нажатия клавиши не было сгенерировано ни одного из 256 стандартных символов ПВМ.

Выполнение в программе функции INKEY$ не приводит к ожиданию системой нажатия какой-либо клавиши; точно так же не происходит высвечивания на экране курсора или символов, соответствующих нажимаемым клавишам. Эта функция позволяет лишь узнать, нажималась ли какая-нибудь клавиша. После выполнения

оператора, содержащего функцию INKEY$, сразу же продолжают выполняться остальные операторы программы, начиная со следующего. Если нужно, чтобы программа останавливалась и не возобновляла работу до момента нажатия пользователем какой-либо клавиши, то необходимо повторять выполнение до тех пор, пока это не произойдет.

Пример

```
list
10 LOCATE ,,1 'Появление на экране курсора
20 K$=INKEY$:IF LEN(K$)=0 THEN 20
     ELSE K%=ASC(K$) 'Ожидание нажатия клавиши
30 IF K%=13 THEN END'<-'клавиша окончания работы
40 IF K%>31 OR (K%<28 AND K%>13) OR
   (K%<8 AND K%>0) THEN PRINT K$; 'Вывод символа
   на экран
50 GOTO 20 'Переход к вводу следующего символа
Ok
```

Приведенная выше программа содержит цикл (строка 20), который будет выполняться неограниченное число раз до тех пор, пока пользователь не нажмет какую-нибудь клавишу. Если это будет клавиша ↵, программа сразу же завершит работу (строка 30). При нажатии любой другой клавиши оператор PRINT выведет на экран дисплея соответствующий символ (строка 40).

◆◆◆Заметим, что в операторе IF-THEN для проверки состояния клавиатуры не используется непосредственно функция INKEY$; вместо этого текущее значение функции предварительно присваивается некоторой переменной, которая затем уже и анализируется оператором IF-THEN. Это позволяет зафиксировать в программе мгновенное состояние клавиатуры. Если же и в операторе IF-THEN, и в операторе PRINT непосредственно использовалась бы функция INKEY$, то ее вхождение в оператор IF-THEN соответствовало бы состоянию клавиатуры в один момент времени, а в оператор PRINT — в другой. В результате значение функции INKEY$ при выполнении оператора IF-THEN почти наверняка отличалось бы от ее значения при выполнении PRINT.

При нажатии некоторых клавиш генерации какого-либо стандартного символа ПВМ не происходит. Функция INKEY$ позволяет идентифицировать 97 таких клавиш. В их число входят клавиши управления курсором, расположенные на малой клавиатуре (**Home**, ←, **End** и т. д.), и функциональные клавиши с нулевым определением программируемого назначения, т. е. имеющие в качестве определения строковое значение нулевой длины.

При нажатии одной из таких 97 клавиш функция INKEY$ принимает двухсимвольное строковое значение, которое называется *расширенным кодом*. Код первого символа всегда равен 0, а код вто-

рого символа определяет, какая из 97 клавиш была нажата. В приложении D перечислены все 97 расширенных кодов вместе с клавишами, которым эти коды соответствуют.

В программе любой из 97 нестандартных символов можно использовать для организации выполнения той или иной команды или определенного специального действия. Для этого клавишам, генерирующим нестандартные символы, можно заранее поставить в соответствие различные определенные действия. Тогда, если в процессе работы программы окажется, что функция INKEY$ принимает двухсимвольное значение, то, анализируя код второго символа этого значения, можно установить, какое действие следует выполнить.

Пример. Использование клавиш управления курсором для выполнения программой действий по перемещению курсора на экране влево, вправо, вверх и вниз.

```
list
10 LOCATE ,,1 'Появление курсора на экране
20 K$=INKEY$:IF LEN(K$)=0 THEN 20 ELSE K%=ASC(K$)
   'Ожидание нажатия клавиши
30 IF K%=13 THEN END'<-'клавиша завершения
40 IF K%>31 OR (K%<28 AND K%>13) OR (K%<8 AND
   K%>0) THEN PRINT K$; 'Вывод на экран
50 IF LEN(K$)<2 THEN 20 ELSE K%=ASC(RIGHT$(K$,1))
   'Проверка, является ли код расширенным
60 IF K%=72 THEN PRINT CHR$(30); 'вверх
61 IF K%=75 THEN PRINT CHR$(29); 'влево
62 IF K%=77 THEN PRINT CHR$(28); 'вправо
63 IF K%=80 THEN PRINT CHR$(31); 'вниз
70 GOTO 10 'Переход к следующему нажатию клавиши
Ok
```

Приведенная выше программа почти в точности совпадает с программой предыдущего примера, только здесь еще дополнительно анализируются случаи, когда функция INKEY$ принимает двухсимвольное значение, и в зависимости от того, равен ли соответствующий расширенный код 72, 75, 77 или 80, выполняется то или иное действие (строки 50—63). Эти коды соответствуют значениям, которые функция INKEY$ принимает при нажатии клавиш управления курсором (приложение D). Таким образом, если будет нажата одна из этих клавиш, программа этот факт обнаружит, и в результате оператором PRINT, содержащим функцию CHR$, будет произведено перемещение курсора в строку, соответствующую нажатой клавише (строки 60—63). Коды, являющиеся аргументами функции CHR$, приведены в Приложении D.

Представление шаблонов входных сообщений

Ни один из операторов и ни одна из функций, управляющих вводом данных с клавиатуры, практически не предоставляет пользователю никакой информации о том, какое количество вводимых символов является допустимым, должно ли быть вводимое значение числовым или оно может содержать любые символы. Подобные сведения, однако, существенно упростили бы работу пользователю, и поэтому следует предусматривать их выдачу в процессе выполнения программы. Для этого в программу можно включить дополнительные операторы, которые непосредственно перед появлением на экране запроса на ввод с клавиатуры будут выдавать на экран шаблон вводимого сообщения. Таким шаблоном может служить просто строка из звездочек или нечто подобное.

Пример. Использование шаблонов.

```
list
10 CLS
190 '——Вывод шаблона
200 LOCATE 12,20,1:PRINT STRING$(10,"*");
210 LOCATE 12,20,1 'Установка курсора для ввода записи
230 K$=INKEY$:IF LEN(K$)=0 THEN 230
    ELSE K%=ASC(K$) ' Ожидание нажатия клавиши
240 IF K%=13 THEN 1500 '<–'клавиша завершения
265 ' Вывод символа на экран и добавление его к вводимому
    сообщению
270 IF K%>31 OR (K%<28 AND K%>13) OR
    (K%<8  AND K%>0) THEN PRINT K$;
    :NTRY$=NTRY$+K$
280 GOTO 230 ' Переход к вводу следующего символа
1500 PRINT:PRINT NTRY$:END
Ok
```

Данный пример аналогичен предыдущему, но здесь в программу введены некоторые усложнения. В начале программы производятся очистка экрана и высвечивание шаблона вводимой записи; шаблон располагается в 12-й строке, начиная с 20-й колонки (строки 10 и 200). Затем восстанавливается прежняя позиция курсора, в результате чего курсор располагается у начала области, предназначенной для ввода сообщения (строка 210). После этого в программе не производится никаких действий до тех пор, пока не будет нажата какая-нибудь клавиша, а как только пользователь это сделает, в программе запоминается значение кода нажатой клавиши (строка 230). По значению этого кода определяется, не была ли нажата клавиша ←┘, и если это действительно так, то на экран выводится все введенное с клавиатуры значение и программа завершает свою работу (строка 240). Если же была нажата любая другая клавиша,

то на экран выводится соответствующий ей символ, который добавляется к вводимому значению (строка 270). При этом, если оказалось, что нажатой клавише соответствует «опасный» символ (вывод которого на экран может испортить все изображение), то в программе такой символ игнорируется, т. е. над ним не производится никаких действий (строка 270). Заметим, что в действительности эта программа не ограничивает количество символов, которые может ввести пользователь, а лишь указывает посредством шаблона, какое число символов считается приемлемым.

Универсальная подпрограмма ввода с клавиатуры

Программа предыдущего примера обладает рядом недостатков. Так, при использовании ее для ввода с клавиатуры пользователь, как и ранее, может испортить изображение на экране, продолжая

```
190 '== Универсальная подпрограмма ввода ======================
200 LOCATE INROW%,INCOL%,1:PRINT STRING$(INLEN%,INTMPL$); 'Вывод шаблона на экран
210 LOCATE INROW%,INCOL%,1 'Установка курсора для ввода записи
220 NTRY$="" 'Обнуление переменной для ввода новой записи
230 K$=INKEY$:IF LEN(K$)=0 THEN 230 ELSE K%=ASC(K$) 'Ожидание нажатия клавиши
240 IF K%=13 THEN 300 'Клавиша <-- завершает ввод записей
250 IF K%=8 THEN 350 'Возврат на одну позицию
260 IF LEN(NTRY$)=INLEN% THEN 230 'Вся запись введена
270 IF K%>31 OR (K%<28 AND K%>13) OR (K%<8 AND K%>0) THEN PRINT K$;
    :NTRY$=NTRY$+K$ 'Вывод символа на экран и добавление к вводимой записи
280 GOTO 230 'Переход к вводу следующего символа
290 '-- Ввод записи завершен; проверка введенного числ. значения по допустимому диапазону
300 NTRY#=VAL(NTRY$)'Преобразование введенной записи в числовое значение
310 IF NTRY#>INMAX# THEN 200 'Слишком велико
320 IF NTRY#<INMIN# THEN 200 'Слишком мало
330 RETURN
340 '-- Возврат на 1 символ ----------------------
350 IF LEN(NTRY$)=0 THEN 230 'Переход к вводу следующего символа, если запись пустая
360 PRINT CHR$(29);LEFT$(INTMPL$,1);CHR$(29);'Замена посл. символа символом шабл.
370 NTRY$=LEFT$(NTRY$,LEN(NTRY$)-1) 'Удаление последнего символа
380 GOTO 230 'Переход к вводу следующего символа
```

Рис. 11.4. Универсальная подпрограмма ввода данных с клавиатуры.

ввод символов за пределами предусмотренной зоны; он не может при этом использовать и клавишу ← для исправления ошибок. Кроме того, приведенная выше программа лишь вводит некоторое строковое значение, а в ряде случаев было бы полезно преобразовывать его в числовое и проверять, лежит ли это числовое значение в заданном диапазоне. Чтобы учесть все указанные недостатки и получить в результате универсальную подпрограмму ввода данных с клавиатуры, достаточно лишь немного изменить предыдущую программу (рис. 11.4).

В этой подпрограмме используются несколько переменных, которым должны быть присвоены определенные значения перед тем,

как подпрограмма будет вызвана. Значением переменной INLEN% является длина вводимого сообщения; значения переменных INROW% и INCOL% указывают позицию на экране дисплея, с которой должна начинаться вводимая строка, а в качестве значения переменной INTMPL$ должен быть задан символ шаблона вводимой записи. В результате работы подпрограммы формируется значение введенной записи (как значение переменной NTRY$) и ее числовое представление (как значение переменной NTRY#).

В начале работы универсальной подпрограммы ввода на экран дисплея, начиная с заданной позиции, выводится шаблон (строка 200), а затем курсор вновь подводится к началу зоны ввода (строка 210). После этого обнуляется переменная NTRY$, в которой накапливается вводимое с клавиатуры значение (строка 220). Далее в подпрограмме не производится никаких действий до тех пор, пока не будет нажата какая-нибудь клавиша. Как только это произойдет, в подпрограмме запоминается значение соответствующего нажатой клавише кода (строка 230). При этом проверяется, не была ли нажата клавиша ↵, и если это так, то подпрограмма переходит к преобразованию введенного сообщения в числовое значение и проверке, лежит ли это числовое значение в заданных пределах (строки 240, 300—320). Если клавиша ↵ не нажималась, то проверяется, не была ли это клавиша ←. В случае отрицательного ответа следует проверка достаточности введенных символов для формирования полной записи (строка 260). Если оказалось, что введено уже достаточное число символов, то в дальнейшем подпрограмма будет игнорировать нажатие любых клавиш, кроме клавиш ← и ↵.

Когда нажатой оказывается клавиша ←, подпрограмма проверяет, содержит ли вводимое сообщение хотя бы один символ (строка 350). Если нет, то произведенное нажатие клавиши игнорируется, поскольку в данном случае операция возврата на одну позицию влево не имеет смысла, и подпрограмма переходит в режим ожидания следующего нажатия какой-нибудь клавиши. В противном случае производится перемещение курсора на одну позицию влево, а символ, стоящий на экране в позиции, соответствующей новому положению курсора, заменяется на символ шаблона вводимой записи, но сам курсор при этом уже не меняет своего положения (строка 360). Кроме того, в этом случае из вводимого сообщения удаляется замененный символ (строка 370), после чего подпрограмма ожидает очередного нажатия клавиши (строка 380).

При нажатии любой другой клавиши на экран выводится соответствующий ей символ, который добавляется к вводимому строковому значению (строка 270). При этом, как и в предыдущей программе, игнорируются все символы, вывод которых на экран может оказать нежелательные воздействия на изображение (строка 270).

После того как пользователь нажимает клавишу ↵, универсаль-

ная подпрограмма ввода выполняет преобразование введенной записи в числовое значение (строка 300). Затем проверяется соответствие этого числового значения диапазону, заданному в вызывающей программе (строки 310 и 320); значение переменной INMAX# является наибольшим допустимым числом, а значение переменной INMIN# — наименьшим. Если оказалось, что введенное числовое значение выходит за указанные пределы, то подпрограмма отбрасывает его и переходит к вводу новой записи.

Многократный ввод с клавиатуры

Программы, в которых с клавиатуры должно вводиться много сообщений, удобнее писать и использовать, если последние организованы некоторым специальным образом. Вместо того чтобы вводить нужные значения последовательно одно за другим, в программе можно предусмотреть выдачу на экран дисплея той или иной формы представления информации, указывающей, какие именно сообщения должны вводиться, и содержащей пустые места, заполняемые пользователем при вводе конкретных значений. В этом случае пользователь всегда будет знать, какие записи он уже ввел, а какие еще предстоит ввести. Стандартная подпрограмма ввода (рис. 11.4) обладает для этого достаточными возможностями.

Разрабатывая формы входных сообщений, выдаваемые на экран, удобно использовать диаграммную бумагу (рис. 11.5). Каждой клетке такой бумаги можно сопоставить одну символьную позицию экрана. В верхней части листа указываются номера столбцов, а сбоку — номера строк сверху вниз. С помощью этих номеров впоследствии идентифицируются любые клетки. Затем на листе записываются названия всех входных сообщений и размечаются клетки для заполнения вводимыми сообщениями. Предварительное конструирование форм на бумаге позволяет сократить общее время, затрачиваемое на программирование: значительно быстрее можно перегруппировать элементы сообщений или изменить названия некоторых из них на бумаге, чем каждый раз заново перепрограммировать разрабатываемую форму.

Как только разработан окончательный вариант формы представления информации, не составляет труда написать программу, которая будет выводить эту форму на экран. Номера строк и столбцов, записанные по краям бумажного листа, можно непосредственно использовать в качестве номеров строк и колонок экрана в операторах LOCATE и в функциях TAB, с помощью которых осуществляется перемещение курсора в нужное положение перед выводом на экран оператором PRINT очередного названия записи. После того как в программе предусмотрены все необходимые операторы LOCATE и PRINT (с включенными в них функциями TAB), остается лишь добавить несколько операторов вызова универсальной

подпрограммы для организации форматированного ввода сообщений.

На рис. 11.6 показана программа, которая выдает на экран форму, приведенную на рис. 11.5, и осуществляет ввод конкретных значений в соответствии с этой формой. Для стыковки данной программы с универсальной подпрограммой ввода данных с клавиату-

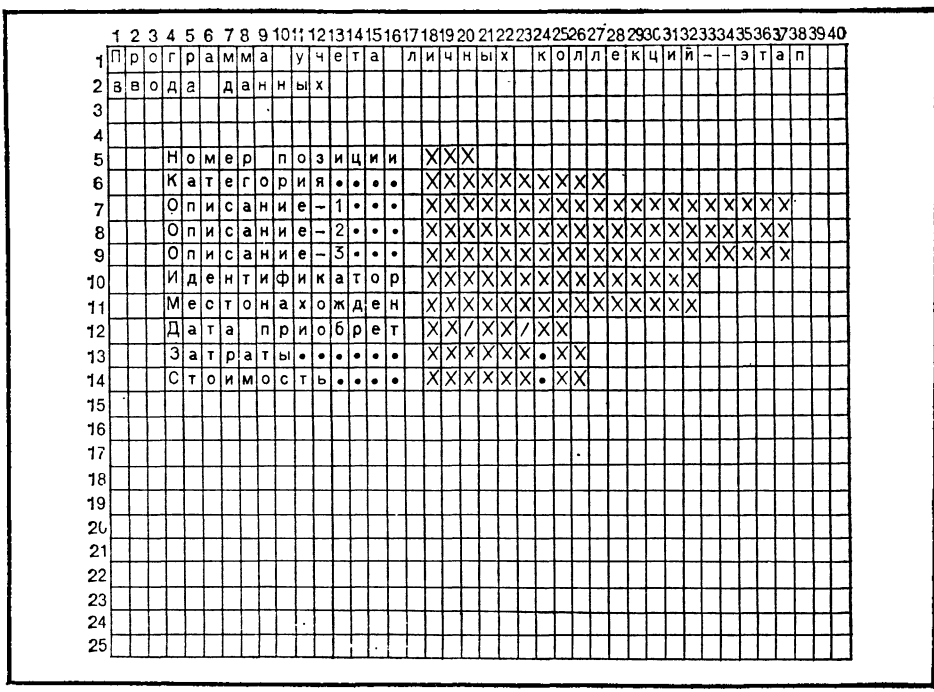


Рис. 11.5. Использование диаграммной бумаги для разработки формы представления информации на экране дисплея.

ры используется оператор CHAIN MERGE (строка 200). Основная программа начинает свою работу с подготовки экрана дисплея к приему данных с клавиатуры (строка 1000) и с выдачи на экран стандартной формы для ввода сообщений (строки 1300—1360). В данной программе названия, присутствующие в форме, содержатся в операторе DATA (строка 7000), хотя возможны и другие способы их хранения в программе. После выдачи формы на экран программа формирует запрос на ввод с клавиатуры всех требуемых записей (строки 1400—1420); операторы, осуществляющие ввод каждого элемента сообщения, оформлены в виде подпрограммы (строки 4000—4800). Такое глубокое структурирование программы на первый взгляд может показаться излишним, однако ниже будут

```
200 CHAIN MERGE "11-4",1000 'Стыковка с подпрограммой ввода с клавиатуры
1000 CLS:KEY OFF:WIDTH 40
1290 '--Вывод на экран формы входчых сообщений----------
1300 CLS
1310 PRINT "Программа учета личных коллекций -этап ввода данных"
1320 RESTORE 7000
1330 FOR ROW%=5 TO 14
1340 READ COL%,LABEL$
1350 LOCATE ROW%,COL%:PRINT LABEL$;STRING$(13-LEN(LABEL$),".");
1360 NEXT
1390 '--Последовательный ввод всех записей----------
1400 FOR N%=1 TO 10
1410 GOSUB 4000
1420 NEXT
1800 REM Здесь должен быть вывод информации в дисковый файл (рис. 16-2)
1890 '--Прдолжать ввод записей?----------
1900 LOCATE 2,1:PRINT "Ввести еще одну запись? (Y/N)";TAB(40)
1910 LOCATE 3,1:PRINT TAB(40) 'Удаление старого текста
1920 INLEN%=1:INMAX#=0:INMIN#=0:INROW%=3:INCOL%=1:INTMPL$="*":GOSUB 200
1930 IF NTRY$<>"N" AND NTRY$<>"n" THEN 1300
2090 '-- Концевая часть программы----------
2100 CLS
2110 PRINT "Программа учета личных коллекций -конец"; TAB(39);
2120 PRINT TAB(5);TIME$;TAB(15);DATE$
2130 END
3990 '== Ввод записи с конкретным N% ==========
4000 LOCATE 2,1,0:PRINT "Введите запись, заполняя шаблон";TAB(40):PRINT
4010 PRINT "Наж.Кл." ;CHR$(17);CHR$(196);CHR$(217);"для завер. ввода ";TAB(40)
4020 INMAX#=10+20:INMIN#=-INMAX#:INROW%=N%+4:INCOL%=18:INTMPL$="*"
4050 ON N% GOTO 4100,4200,4300,4300,4300,4400,4500,4600,4700,4800
4090 '-- Номер позиции ----------
4100 INLEN%=3:GOSUB 200:ITMNR$=NTRY$
4110 '-- Удаление неиспользованной части шаблона
4120 PRINT SPACE$(INLEN%-LEN(NTRY$));:RETURN
4190 '-- Категория----------
4200 INLEN%=10:GOSUB 200:CTGRY$=NTRY$:GOTO 4120
4290 '-- Описание 1,2 и 3----------
4300 INLEN%=20:GOSUB 200:DSCR$(N%-2)=NTRY$:GOTO 4120
4390 '--Идентификатор ----------
4400 INLEN%=15:GOSUB 200:IDNBR$=NTRY$:GOTO 4120
```

```
4490 '-- Местонахожден.----------
4500 INLEN%=15:GOSUB 200:LCN$=NTRY$:GOTO 4120
4590 '--Дата приобрет.----------
4600 INLEN%=8:GOSUB 200:ACQRD$=NTRY$:GOTO 4120
4690 '--Затраты ----------
4700 INMAX#=999999.99#:INMIN#=0:INLEN%=9:GOSUB 200:COST#=NTRY#
4710 '--Повторный вывод числового значения введенной записи
4720 LOCATE INROW%,INCOL%:PRINT USING "######.##";NTRY#;:RETURN
4790 '- Стоимость ----------
4800 INMAX#=999999.99#:INMIN#=0:INLEN%=9:GOSUB 200:VALU#=NTRY#:GOTO 4720
6980 '--- Наименование записей и шаблоны----------
6990 ' для формы ввода
7000 DATA 4,Номер позиц.,4,Категор.,4, Описание -1,4,  Описание -2,4,
        Описание -3,4, Идентиф.4, Местонах.,4,Дата приобрет.,4, Затраты,4,Стоим.
```

Рис. 11.6. Программа форматированного ввода данных с клавиатуры с использованием формы, приведенной на рис 11.5, и универсальная подпрограмма ввода (рис. 11.4).

приведены примеры, показывающие, что оно вполне оправдано. Введя все требуемые сообщения, пользователь должен решить, будет ли он заполнять еще одну форму или нет; в соответствии с этим программа повторит или завершит свою работу (строки 1900—1930). Полезно обратить внимание на то, как с помощью функции TAB в программе удаляются с экрана старые сообщения и записи, чтобы они не мешали выводу новых (строки 1900 и 1910).

При конструировании формы представления выводимых на экран данных следует учитывать, что в нее могут входить не только вводимые сообщения и их названия. В ряде случаев может потребоваться еще и вывод некоторой вспомогательной информации или результатов каких-либо производимых в программе вычислений. При разработке формы рекомендуется оставлять три или четыре строки незаполненными на случай, если потребуется вводить в программу какую-то дополнительную информацию, выводить данные о состоянии программы или ее название.

Ошибки при вводе

Программа, в которой предполагается, что пользователь, вводя с клавиатуры различные значения, не будет допускать ошибок, просто нежизнеспособна. Ошибки при вводе практически неизбежны, и потому всякая программа должна включать в себя средства для их обработки.

Процедура обработки каждой ошибки состоит из двух этапов: выявления ошибки и оповещения о ней. Универсальная подпрограмма ввода предусматривает выявление ошибок, допускаемых при вводе данных с клавиатуры, с помощью функции INKEY$ (рис. 11.4). Однако эта подпрограмма не предоставляет пользователю почти никакой информации о возникающих ошибках, что очень неудобно. Например, если в процессе выполнения подпрограммы обнаруживается, что была нажата неразрешенная клавиша, ее дальнейшие действия аналогичны тем, которые выполняются в случае, когда не нажимается вообще никакая клавиша. В результате, если при нажатии некоторой клавиши не ввелся никакой символ, пользователь не может различить, произошло ли это по причине использования недопустимой клавиши или из-за неисправности клавиатуры.

Оповещение об ошибках ввода

Один из способов оповещения об ошибке состоит в том, чтобы каждый раз при ее обнаружении включалось встроенное в ПВМ звуковое сигнальное устройство. Этот способ реализуется с помощью оператора BEEP. Все, что нужно сделать для его вставления в

универсальную подпрограмму ввода,— это добавить в строку 270 подпрограммы конструкцию ELSE, как показано ниже:

```
270 IF K%>31 OR (K%<28 AND K%>13) OR
    (K%<8 AND K%>0) THEN PRINT K$;:
    NTRY$=NTRY$ + K$ ELSE BEEP
```

При выполнении такой видоизмененной строки допустимый символ будет по-прежнему выводиться на экран и добавляться ко всей вводимой записи, зато нажатие на клавишу, соответствующую

```
270 IF K%>31 OR (K%<28 AND K%>13) OR (K%<8 AND K%>0) THEN PRINT K$;
   :NTRY$=NTRY$+K$ ELSE BEEP 'Соотв. символ выводится на экран, случ. нажат. игнорируется
310 IF NTRY#>INMAX# THEN ERM$="Уменьшить": GOTO 400 'Слишком велико
320 IF NTRY#<INMIN# THEN ERM$="Увеличить": GOTO 400'Слишком мало
390 '--- Высвечивание на экране сообщения об ошибке -----
400 FOR X1%=1 TO 3
410 LOCATE INROW%,INCOL%,0:PRINT LEFT$(ERM$,INLEN%);'Вывод сообщения
420 BEEP
430 FOR X2%=1 TO 500:NEXT 'Пауза
440 LOCATE INROW%,INCOL%,0:PRINT SPACE$(INLEN%);'Удаление сообщения с экрана
450 FOR X2%=1 TO 500:NEXT 'Пауза
460 NEXT X1%
470 GOTO 200 'Повторение ввода
```

Рис. 11.7. Изменения к универсальной программе ввода данных (рис. 11.4), позволяющие уведомлять пользователя об ошибках ввода.

недопустимому символу, будет не только игнорироваться, но и вызывать звуковой сигнал.

Универсальную подпрограмму ввода можно было бы усовершенствовать, если предусмотреть в процессе работы выдачу пользователю сообщения о том, является ли введенное им значение слишком большим или слишком малым. В том виде, в каком она показана на рис. 11.4, эта подпрограмма не обеспечивает пользователя вообще никакой информацией о причинах отклонения ею конкретного вводимого сообщения. Для устранения этого недостатка достаточно добавить в подпрограмму несколько новых строк и изменить несколько старых. Эти изменения приведены на рис. 11.7.

Поясним, как работает видоизмененная часть подпрограммы. Если оказалось, что введенное значение слишком велико или слишком мало, то с переменной ERM$ связывается текст сообщения об ошибке, выводимого на экран (строки 310 и 320). Затем это сообщение трижды высвечивается на экране на месте введенной записи; для этого используется цикл FOR/NEXT (строки 400—460). Сначала выводится часть сообщения об ошибке, поместившаяся в зоне, отведенной для вводимого сообщения (строка 410), затем подается звуковой сигнал (строка 420) и с помощью цикла FOR/NEXT (строка 430) организуется двухсекундная пауза и сообщение

удаляется с экрана (строка 440). Еще одна такая же пауза выдерживается перед повторением операторов всего внешнего цикла (строка 450). После третьего повторения этого цикла управление передается на начало универсальной подпрограммы ввода для приема повторно вводимого значения (строка 470).

Средства исправления ошибок ввода

Существует целый ряд ошибок, которые в принципе нельзя выявить с помощью программы. Например, при вводе сообщений пользователь может делать орфографические ошибки, указывать неверные числа и допускать другие неточности, не нарушая при этом правдоподобности сообщения, которое в результате будет восприниматься программой как правильное. Ошибки подобного рода способен обнаружить только сам пользователь, однако программа может облегчить ему эту задачу.

Полезно, в частности, предоставить пользователю возможность контроля вводимых сообщений и их исправления по окончании просмотра. Если бы, например, в программе каждой вводимой с клавиатуры записи присваивался уникальный номер, то с помощью этих номеров пользователь мог бы идентифицировать ту из них, которую он хочет исправить. Приведенная выше программа форматированного ввода данных (рис. 11.5 и рис. 11.6) не предусматривает указанных возможностей.

```
1590 '--Вывод номеров записей на экран ----------------------
1600 FOR X1%=1 TO 10
1610 LOCATE X1%+4,1:PRINT USING "##)";X1%
1620 NEXT X1%
1630 '-- Ввод номера изменяемой записи----------------------
1640 LOCATE 2,1:PRINT "Введите номер записи (если изм.нет, введите 99)";TAB(40)
1650 LOCATE 3,1:PRINT TAB(40)
1660 INLEN%=2:INMAX#=99:INMIN#=1:INROW%=3:INCOL%=1:INTMPL$="#":GOSUB 200
1670 IF NTRY#>10 THEN 1800
1680 N%=NTRY#:GOSUB 4000
1690 GOTO 1640
```

Рис. 11.8. Дополнительные строки к программе форматированного ввода данных (рис. 11.6), позволяющие изменять введенные записи, идентифицируемые по их номерам.

На рис. 11.8 приведены программные строки, которые следует добавить в программу рис. 11.6 для того, чтобы пользователь мог изменять уже введенные им данные. При выполнении этого нового модуля программы сначала на экран рядом с каждым указанным в форме названием записи выводится ее идентификационный номер (строки 1600—1620), а затем пользователь осуществляет ввод номера записи, которая будет изменена (строки 1640—1660). После этого вызывается подпрограмма ввода элементов сообщения, с по-

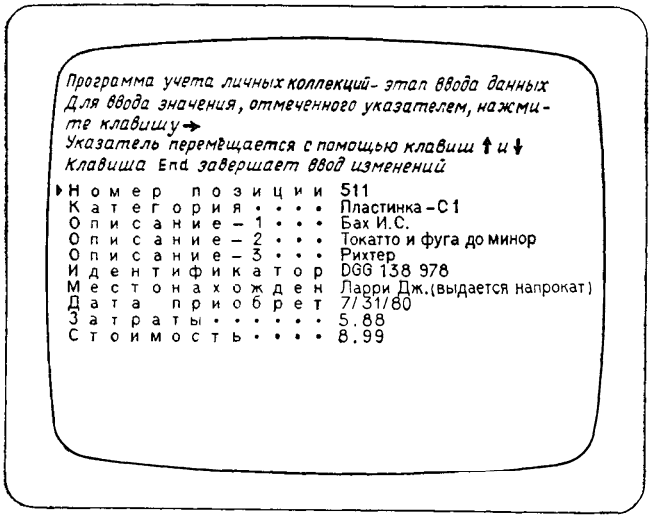


Рис. 11.9. Использование указателя для отметки изменяемой записи. (На рисунке написано «Токатто»; следует читать «Токката».)

мощью которой на место старой записи вводится новая (строки 1680). Все описанные действия повторяются неограниченное число раз до тех пор, пока пользователь не введет в качестве номера изменяемой записи число 99, после чего ввод изменений завершается (строки 1670 и 1690).

Сообщая номер изменяемой записи, пользователь вынужден выполнять работу, которую можно было бы сделать в самой программе. В большинстве случаев было бы удобнее, не указывая никаких

```
1590 '--Внесение изменений в запись----------
1600 N%=1 ' Установка начального положения указателя на первую запись
1610 LOCATE 2,1:PRINT"Для ввода знач., отмеч. указат., нажм. клав.";CHR$(26);
1620 PRINT "Указат. перем. с пом. клав."; CHR$(24); "и"; CHR$(25); End заверш. ввод измен.
1630 LOCATE N%+4,2,0:PRINT CHR$(16); 'Высвечивание указателя
1640 K$=INKEY$:IF LEN(K$)=0 THEN 1640 'Ожидание нажатия клавиши
1650 PRINT CHR$(29);" "; 'Гашение указателя
1660 '-- Игнорирование незначащих клавиш
1670 IF LEN(K$)<>2 THEN BEEP:GOTO 1630 ELSE K%=ASC(RIGHT$(K$,1))
1680 IF K%=77 THEN GOSUB 4000:GOTO 1610 ' Стрелка вправо
1690 IF K%=79 THEN 1800 ' Клавиша End
1700 IF K%=72 AND N%>1 THEN N%=N%-1:GOTO 1630 ' Стрелка вверх
1710 IF K%=80 AND N%<10 THEN N%=N%+1:GOTO 1630 ' Стрелка вниз
1720 BEEP:GOTO 1630 'Бессмысленное нажатие клавиши
```

Рис. 11.10. Дополнительные строки к программе форматированного ввода данных (рис. 11.6), позволяющие изменять введенные записи, идентифицируемые с помощью указателя (эти строки вставляются в программу вместо приведенных на рис. 11.8).

номеров, просто переходить от одной записи к другой, нажимая клавиши перемещения курсора ↑ и ↓, а для идентификации изменяемых записей вместо номеров использовать специальный указатель, который высвечивается на экране рядом с изменяемой записью (рис. 11.9).

На рис. 11.10 приведена программа, реализующая описанный способ внесения изменений. При выполнении этой программы сначала на экран слева от первой записи выводится указатель (строки 1600—1630), а затем программа переходит в состояние ожидания нажатия клавиши (строка 1640). Как только это произойдет, программа удаляет с экрана указатель (строка 1650) и начинает анализировать код, соответствующий нажатой клавише (строки 1670—1710). Дальнейшие действия программы определяются значением этого кода: если пользователь нажал клавишу →, то программа запрашивает повторный ввод записи, соответствующей текущему положению указателя (строка 1680); если же была нажата клавиша **End**, программа завершает ввод изменений (строка 1690). Нажатие клавиши ↑ или ↓ приводит к перемещению указателя вверх или вниз по позициям списка (строка 1700 и 1710); любые другие клавиши программой игнорируются (строки 1670 и 1720).

Глава 12

ФАЙЛЫ ДАННЫХ НА ДИСКАХ

В тех случаях, когда программа обрабатывает достаточно большой объем данных, последние должны организовываться в файлы и храниться вне динамической памяти персональной ЭВМ. Наиболее эффективным средством организации внешнего хранения данных являются диски, и большинство вычислительных систем, в которых используются ПВМ, имеет по меньшей мере один дисковод.

В настоящей главе показано, как с помощью программы, написанной на языке Бэйсик, осуществляется запоминание и поиск данных в дисковых файлах; при этом подразумевается, что читатель усвоил материал, изложенный в гл. 3. Операторы и функции, о которых пойдет речь, относятся только к дисковому и расширенному Бэйсику.

Структура файлов данных

Для описания файловой структуры данных требуется привлечение широкого круга понятий, в том числе и понятия дисковый файл (гл. 3). Каждый файл подразделяется на одну или более *записей*, подобных карточкам в картотечном ящике. Запись обычно содержит несколько значений данных, точно так, как карточки содержат отдельные позиции. Значение каждого элемента информационной записи называется *полем*.

Обычно файл организуется так, что все его записи имеют одинаковую конфигурацию. Количество полей, порядок их расположения и длина не изменяются при переходе от одной записи к другой. Единственное, чем отличаются записи,— это значениями полей, и потому компоновка файла определяется двумя факторами: конфигурацией и количеством содержащихся в нем записей.

Таким образом, перечисление полей одной записи дает адекватное описание структуры файла. Программисты часто составляют перечень полей данных, пользуясь стандартным типографским бланком, и получившийся в результате документ называют *макетом файла* (рис. 12.1). Обычно в этом документе указывается имя файла, объявляется количество содержащихся в нем записей и перечисляются по порядку поля. Для каждого поля приводятся его описание и длина, а также имя переменной, обычно используемое для задания значения поля в программе, написанной на Бэйсике.

Один дисковый накопитель обеспечивает доступ только к определенному объему данных, поэтому целесообразно заранее вычислять, останется ли достаточно места на конкретном дискете для вновь формируемого файла. Для этого суммируют длины всех полей одной

записи и получившуюся сумму умножают на количество записей в файле. Проделав подобную операцию для каждого файла, планируемого на диске, сравнивают суммарную потребность с общей емкостью диска. Последнюю можно определить с помощью описанной в гл. 3 команды CHKDSK, имеющейся в составе операционной системы ДОС ПВМ. Помимо области для файлов данных необходим

Макет файла

Имя файла ITEM.DAT	Длина записи 129	Число записей
Описание Файл, содержащий сведения о личных коллекциях,— последовательный доступ		

Имя переменной	Назначение полей	Максимальный размер	Примечания
ITMNR$	Номер позиции	3	
CTGRY$	Категория предметов	10	
DSCR$(1)	Описание — строка 1	20	
DSCR$(2)	Описание — строка 2	20	
DSCR$(3)	Описание — строка 3	20	
IDNBR$	Идентификационный номер	15	
LCN$	Местонахождение	15	
ACQRD$	Дата приобретения	8	
COST#	Исходная цена (затраты)	9	
VALU#	Стоимость	9	

Рис. 12.1. Макет файла с последовательным доступом, содержащего сведения о личных коллекциях.

достаточный участок памяти для файлов программ, которые предполагается разместить на том же диске.

Иногда для размещения слишком большого файла в дисковой памяти достаточно уменьшить длину одного или двух полей данных. Если же это не помогает, то файл разделяют на несколько частей и распределяют эти меньшие файлы между несколькими дисками. Можно разделить дисковый файл на несколько томов, поместив часть его записей на один диск, а часть — на другой, или даже довести разбиение до отдельных записей, поместив одни поля в файл на одном диске, а другие — в файл на другом диске.

Последовательный и произвольный доступ

В Бэйсике ПВМ реализуются два способа отыскания нужных записей. Простейший из них называется *последовательным доступом*, поскольку программа всегда должна начинать поиск с начала файла и проверять по очереди каждую запись до тех пор, пока не будет найдена требуемая. Альтернативный способ позволяет программе обращаться к записям по номеру в любом порядке. Такой способ поиска называется *произвольным доступом*.

Любой файл возможно организовать так, чтобы можно было использовать либо последовательный, либо произвольный доступ, но не оба способа одновременно. Каждый способ имеет свои «за» и «против». Последовательный доступ намного проще для программирования, и при его реализации требуется меньше дисковой памяти, но может понадобиться больше времени для поиска записи, находящейся в конце длинного файла. Кроме того, обновление существующих записей в файлах с последовательным доступом трудноосуществимо, а иногда и просто невозможно. Файлы с произвольным доступом требуют более сложного программирования и занимают обычно больше места на диске, но очень легко поддаются обновлению, и поиск любых записей может осуществляться с одинаковой скоростью.

В последовательном файле длина записи непосредственно зависит от величины каждого поля, а поскольку длины полей в каждой записи различны, то и длины записей также различны. В отличие от последовательного файла, длина записи в файле с произвольным доступом постоянна. Длина записи в файле с произвольным доступом определяется при его создании, и каждому полю выделяется внутри записи строго определенное место. Возможные значения полей должны умещаться в этом заранее отведенном объеме памяти. Если это не продумано заблаговременно, то при выполнении программы чрезмерно большие значения подвергаются усечению, а малые дополняются пустующими ячейками. В целях минимизации нежелательного усечения размер каждого поля обычно выбирается достаточно большим, чтобы вместить самую большую возможную величину.

Использование дисковых файлов данных

Для считывания конкретного поля записи программа должна сначала открыть файл, затем найти соответствующую запись, переслать эту запись с диска в динамическую память и, наконец, выделить требуемое поле в виде значения переменной. Аналогично производится запись информации в файл с той лишь разницей, что в этом случае значения данных пересылаются программой из динамической памяти в дисковый файл.

Для уменьшения числа обращений к диску в Бэйсике ПВМ пересылка данных с диска и на диск производится блоками, а не по одному элементу за каждое обращение. С этой целью часть динамической памяти отводится специально для размещения пересылаемых данных; эти участки запоминающего устройства (ЗУ) называются *буферами файлов*; каждый активный файл имеет свой собственный буфер.

В Бэйсике ПВМ управление буферами осуществляется автоматически. Однако, если программа закончит запись в файл, а в буфере останутся некоторые не записанные на диск данные, программа должна тем или иным способом все же завершить перепись содержимого буфера. Сделать это можно с помощью команды закрытия файла. Одновременно по этой команде в справочник диска вносятся изменения, касающиеся размеров файла и других статистических данных, которые в нем фиксируются. Следовательно, когда программа заканчивает работу с файлом, она должна его закрыть во избежание потери части содержимого файла.

Имена и номера файлов

Файлы данных на диске распознаются по стандартным именам (гл. 3, рис. 3.6 и табл. 3.1). Тем не менее программы обращаются к файлам данных главным образом по номерам. Оператор OPEN соотносит имя файла с его номером:

```
1050 OPEN "B:ITEM.DAT" AS #1
```

Имя файла может быть указано строковой константой, переменной или выражением. Префикс, определяющий диск, требуется тогда, когда файл не находится на диске, принимаемом по умолчанию.

Номер файла может быть указан числовой константой, переменной или выражением; обычно допускаются только номера 1, 2 и 3. Каждый номер идентифицирует только один файл, поэтому программа может открыть одновременно максимум три файла.

Следует заметить, что номера файлов и номера устройств (гл. 10) — это одно и то же, поскольку в действительности термин «номер файла» часто относится к номеру устройства. При использовании операторов PRINT# или PRINT# USING для работы с печатающим устройством и экраном дисплея программа может открывать меньше файлов, так как в таких случаях она должна «расходовать» один номер на каждое открываемое устройство или файл. Если, например, открываются два устройства (скажем, печатающее устройство и экран дисплея), программа способна открыть одновременно с этим только один файл данных. В случае применения операторов PRINT, PRINT USING, LPRINT и LPRINT USING номера файлов и устройств не указываются, благодаря чему эти

операторы могут использоваться в программе без каких-либо ограничений, даже когда все три номера присвоены файлам данных.

Оператор CLOSE освобождает номер файла/устройства для повторного использования, если следующим по порядку идет оператор OPEN.

Пример

```
1100 OPEN "LPT1:" AS #1
7510 CLOSE #1
7520 OPEN "A:TAXRATE.DAT" AS #1
```

Оператор CLOSE, содержащий только один номер (строка 7510), закрывает именно тот определенный файл или устройство, к которому этот номер относится. Чтобы закрыть сразу несколько файлов одним оператором CLOSE, достаточно указать в нем несколько номеров, отделенных друг от друга запятыми, например

```
2140 CLOSE #1, #3
```

Простой оператор CLOSE, в котором нет ни одного номера файла, будет закрывать все уже открытые файлы и устройства. Несколько других операторов языка Бэйсик, включая END, CHAIN (но не CHAIN MERGE), LOAD, NEW, RUN и SYSTEM, выполняют ту же функцию. Однако большинство программистов предпочитают иметь гарантированный точный результат работы оператора CLOSE и поэтому указывают в нем номера файлов в явном виде.

Последовательные дисковые файлы

Программа на языке Бэйсик способна как создавать новые файлы с последовательным доступом для запоминания в них значений нужных данных, так и записывать данные в конец уже существующего файла или извлекать данные из него. В файле с последовательным доступом записи должны считываться или записываться последовательно. Для отыскания какой-либо величины, находящейся в середине файла, программа должна произвести считывание всех данных, расположенных впереди. Новые данные могут быть записаны только в конец последовательного файла. Это значит, что не существует никакого способа считать элемент данных из файла с последовательным доступом, изменить считанное значение и снова записать его в файл на то же самое место.

При работе с последовательными файлами буфера используются самым простым способом: пересылаемые в файл данные накапливаются в буфере файла, а по заполнении буфера его содержимое сразу целиком переписывается в дисковый файл; при этом буфер файла очищается для следующей порции выходных данных. При чтении файла соответствующие данные поступают из того же самого буфера, а не считываются непосредствен-

но с поверхности диска. Как только все необходимые данные считаны из буфера, средства Бэйсика обеспечивают повторное его заполнение информацией из дискового файла.

Открытие и закрытие файлов с последовательным доступом

Кроме присвоения поименованному файлу определенного номера оператор OPEN устанавливает разрешенный режим доступа к данным, хранящимся в этом файле. При использовании последовательных файлов программа может записывать данные с начала нового файла, добавлять их в конец уже существующего или считывать данные из файла. В табл. 12.1 приведены форматы оператора OPEN, обеспечивающие выбор того или иного режима доступа.

Таблица 12.1. **Модификации оператора OPEN для файлов данных**

Режим доступа	Образцы форматов оператора [1])
Запись в начало нового файла с последовательным доступом	1050 OPEN "B:ITEM.DAT" FOR OUTPUT AS#1
Запись в конец существующего файла с последовательным доступом	1835 OPEN "ADDRESS.DAT" FOR APPEND AS#3
Чтение с начала существующего файла с последовательным доступом	2320 OPEN "A:ITEM.DAT" FOR INPUT AS#2
Запись или чтение файла с произвольным доступом	1040 OPEN "ITEM.DAT" AS#1 LEN = 126

[1]) Указанные номера строк, номера файлов и длина записи являются произвольными и взяты только для иллюстрации.

Оператор OPEN с конструкцией FOR OUTPUT создает новый дисковой файл с заданным именем. Если файл с таким именем уже есть, то средства Бэйсика ПВМ автоматически уничтожают его, создавая новый файл с тем же именем. Оператор OPEN-FOR APPEND обеспечивает поиск названного файла среди уже существующих, с тем чтобы добавить новые записи в его конец, но если требуемого файла найти не удается, то автоматически создается новый файл с заданным именем. Модификация оператора с конструкцией FOR INPUT реализует поиск файла данных с конкретным именем, отсутствие которого означает ошибку.

Можно открыть файл с последовательным доступом в режиме записи, используя один номер файла, и в режиме чтения, используя другой. Однако каждому номеру файла будет при этом соответствовать свой буфер, никак не связанный с другими. Поэтому для изменения режима доступа к файлу последний необходимо закрыть,

а затем вновь открыть. Вместе с тем файлы, отличающиеся друг от друга режимами доступа, могут открываться одновременно.

Запись и чтение файлов с последовательным доступом

Операторы PRINT# и PRINT#USING реализуют запись значений в дисковый файл, а операторы INPUT# или INPUT # LINE — считывание этих значений и присваивание их переменным.

Пример

```
List
10 OPEN "SAMPLE.DAT" FOR OUTPUT AS #1
20 PRINT#1, 123 'Запись значения в дисковый файл #1
30 CLOSE #1
40 OPEN "SAMPLE.DAT" FOR INPUT AS #1
50 INPUT#1, A 'Считывание значения из дискового файла #1
60 PRINT "Первое значение в файле: ";A
70 CLOSE #1
Ok
run
Первое значение в файле: 123
Ok
```

Операторы PRINT# форматируют выходные данные совершенно одинаково независимо то того, какое устройство вывода используется, и всегда добавляют пробел после каждого числового значения. В случае нечисловых величин при наличии после оператора PRINT точки с запятой выдаваемые соседние значения записываются слитно, а при наличии запятой между ними вставляется дополнительный пробел. Дополнительные пробелы бесполезно расходуют пространство дисковой памяти, и поэтому предпочтительнее использовать точки с запятой.

Действие шаблонов оператора PRINT#USING одинаково при работе с дисковыми файлами, экраном дисплея и печатающим устройством (табл. 10.1). Однако следует весьма осторожно употреблять префиксы, обозначающие денежные единицы ($$,** и **$), так как они вызывают форматирование числовых величин как нечисловых. По этой причине числовые величины, снабженные в дисковом файле такими префиксами, не могут быть считаны оттуда в их исходном виде.

Пример

```
List
10 OPEN "SAMPLE.DAT" FOR OUTPUT AS #2
20 PRINT#2, USING "$$ ###.##" ;99.50
30 CLOSE # 2
40 OPEN "SAMPLE.DAT" FOR INPUT AS # 2
50 INPUT#2, A
```

```
60 PRINT "Первое значение в файле: " ;A
70 CLOSE #2
Ok
run
Первое значение в файле: 0
Ok
```

В этой программе оператор PRINT# USING (строка 20) пересылает символы "$99.50" (без кавычек) в файл SAMPLE.DAT. Оператор INPUT # реализует считывание этих символов как числовой величины, однако знак $ превращает ее в нечисловую, вследствие чего ей приписывается значение, равное нулю.

Отсутствие завершающей точки с запятой в операторе PRINT# или PRINT # USING приводит к появлению символа возврата каретки за последним указанным в операторе значением. Завершающая точка с запятой обеспечивает подавление этого символа.

Разделение значений в файлах с последовательным доступом

Для того чтобы оператором INPUT# правильно реализовывалось считывание значений из дискового файла, эти значения должны быть в файле отделены друг от друга, и, следовательно, операторы PRINT# и PRINT# USING при записи данных в дисковые файлы должны добавлять к записываемым значениям разделительные знаки. При отсутствии такого разграничения оператор INPUT# обрабатывает соседние значения как одно целостное.

Пример

```
List
10 OPEN "TEST.DAT" FOR OUTPUT AS #3
20 PRINT#3, "Кремний";14;28.0855
30 CLOSE #3
40 OPEN "TEST.DAT" FOR INPUT AS #3
50 FOR K%=1 TO 3:INPUT#3, A$
60 PRINT "Значение номер";K%; ":";A$
70 NEXT K%:CLOSE #3
Ok
run
Значение номер 1: Кремний 14 28.0855
Input past end in 50
Ok
```

При выполнении оператора PRINT# (строка 20) в файл записываются три различных значения, но поскольку в операторе они разделяются точками с запятой, в файл они будут заноситься без разделителей. Когда оператор INPUT# реализует считывание этих значений (строка 50), они все сливаются в одно.

Таким образом, в файле не будет существовать ни второго, ни третьего значений, поэтому при попытке чтения программой какой-либо информации за последним значением в файле фиксируется ошибка "Считывание за пределами последнего значения".

Разделителем для строковых данных может быть запятая или символ возврата каретки. Те же самые знаки или пробелы служат для разграничения числовых величин. Существует несколько способов гарантированной расстановки разделителей между отдельными значениями. Один из самых простых состоит в том, чтобы присвоить значение «запятая» строковой переменной и записывать ее после каждого значения в операторе PRINT#.

Пример

```
 5 D$= ","
20 PRINT#3, "Кремний" ;D$;14;D$;28.0855
```

Оператор INPUT# интерпретирует каждую встреченную запятую как признак конца значения. Единственным исключением из этого правила является запятая, заключенная в кавычки, внутри файла. Иначе говоря, строковая величина в дисковом файле, могущая содержать запятую, должна быть записана в кавычках; оператор INPUT# обеспечивает устранение этих кавычек при считывании.

Пример

```
List
10 OPEN "TEST.DAT" FOR OUTPUT AS #1
20 PRINT#1, CHR$(34); "Манагуа, Никарагуа" ;
   CHR$(34)
30 CLOSE #1
40 OPEN "TEST.DAT" FOR INPUT AS #1
50 INPUT#1, A$
60 PRINT "Первое значение в файле: ";A$
70 CLOSE #1
Ok
run
Первое значение в файле: Манагуа, Никарагуа
Ok
```

Сходная проблема возникает при использовании оператора PRINT# USING, когда запятая присутствует в шаблоне числовых величин. Одна числовая величина, записанная по такому шаблону, будет превращаться в несколько значений в дисковом файле.

Пример

```
List
10 OPEN "SAMPLE.DAT" FOR OUTPUT AS #2
20 PRINT#2, USING " #,##########.## ";
   123456789
```

```
30 CLOSE #2
40 OPEN "SAMPLE.DAT" FOR INPUT AS #2
50 INPUT#2, A
60 PRINT "Первое значение в файле: ";A
70 CLOSE #2
Ok
run
Первое значение в файле: 123
Ok
```

Оператор LINE INPUT#, считывающий одиночное строковое значение, позволяет решить проблему запятой в шаблоне, поскольку в данном случае в качестве разделителей распознаются символы возврата каретки, а запятые представляют собой одно из строковых значений.

Пример

```
List
10 OPEN "SAMPLE.DAT" FOR OUTPUT AS #3
20 PRINT#3, "Стайн, Фрэнк Н."
30 CLOSE #3
40 OPEN "SAMPLE.DAT" FOR INPUT AS #1
50 LINE INPUT#1, A$
60 PRINT "Первое значение в файле: ";A$
70 CLOSE #1
Ok
run
Первое значение в файле: Стайн, Фрэнк Н.
Ok
```

Распознавание конца файла

Зафиксированная при выполнении операторов INPUT или INPUT # LINE попытка продолжить считывание информации за последней записью файла приводит к возникновению ошибки. Во избежание этого программа должна каким-то образом распознавать, что последнее значение считано, и прекратить дальнейшее считывание. Проще всего это реализуется с помощью операторов WHILE или IF-THEN, содержащих функцию EOF (конец файла):

```
1850 IF NOT EOF(1) THEN 1700 ELSE CLOSE #2
```

Число в круглых скобках указывает номер файла. Функция EOF принимает значение «Ложь» (число 0) до тех пор, пока в файле остаются какие-то значения, но как только последнее значение считано, она приобретает значение «Истина» (число —1).

Использование файлов данных с последовательным доступом

Последовательный доступ пригоден для большинства файлов данных, хотя для периодически обновляемых файлов он менее

```
990 '--Открытие файла ----------
1000 CLS:KEY OFF:WIDTH 40
1020 PRINT " Программа учета личных коллекций – этап ввода"
1030 PRINT "Стереть существующий файл? (Y/N)"
1040 INPUT;"",NTRY$
1050 IF NTRY$="Y" OR NTRY$="y" THEN OPEN "ITEM.DAT" FOR OUTPUT
    AS #1 ELSE OPEN "ITEM.DAT" FOR APPEND AS #1
1060 D$=","  Разграничивающая запятая при вводе данных
1390 '-- Последовательный ввод всех позиций ----------
1400 CLS
1410 PRINT " Программа учета личных коллекций – этап ввода"
1420 LOCATE 5,1
1430 INPUT "Номер позиц.";ITMNR$
1440 INPUT "Категория" ;CTGRY$
1450 INPUT " Описание  -1";DSCR$(1)
1460 INPUT " Описание  -2";DSCR$(2)
1470 INPUT " Описание  -3";DSCR$(3)
1480 INPUT "Идентификатор";IDNBR$
1490 INPUT "Местонахожден.";LCN$
1500 INPUT "Дата приобрет.";ACORD$
1510 INPUT "Затраты";COST#
1520 INPUT "Стоимость";VALU#
1790 '-- Занесение одиночной записи в файл данных ----------
1800 PRINT#1,ITMNR$;D$;CTGRY$;D$;DSCR$(1);D$;DSCR$(2);D$;DSCR$(3);D$;
1810 PRINT#1,IDNBR$;D$;LCN$;D$;ACQRD$;D$;COST#;D$;VALU#
1890 '--Продолжать ли ввод сообщений?----------
1900 LOCATE 2,1
1910 PRINT "Вводить ли следующие позиции? (Y/N)"
1920 INPUT;"",NTRY$
1930 IF NTRY$<>"N" OR NTRY$<>"n" THEN 1400
1940 CLOSE #1 'На данный момент сообщений больше нет
1990 '--Нужен ли просмотр файла?----------
2000 CLS
2010 PRINT " Программа учета личных коллекций – этап просмотра файла"
2020 PRINT " Нужен ли просмотр файла? (Y/N)"
2030 INPUT;"",NTRY$
2040 IF NTRY$="y" OR NTRY$="Y" THEN CHAIN"12-3",1000,DELETE 1000-7000
2090 '--Конец программы ----------
2100 CLS
2110 PRINT"Программа учета личных коллекций– работа окончена";TAB(39)
2120 PRINT TAB(5);TIME$;TAB(15);DATE$
2130 END
7000 REM  Конец блока (см. строку 2040)
```

Рис. 12.2. Программа создания файла, содержащего сведения о личных коллекциях: запись данных в файл с последовательным доступом (может быть объединена с программой рис. 11.6; пояснения см. в тексте).

удобен, чем прямой доступ. Последовательный доступ хорош там, где содержимое файлов не меняется или новые записи заносятся в них от случая к случаю. Например, маловероятно, чтобы часто изменялся файл простой программы, формирующей сведения об основных предметах личной коллекции (рис. 12.1 и 12.2). На рис. 12.3 приведена программа, считывающая записи, созданные программой, которая организует хранение соответствующих записей в дисковом файле с последовательным доступом (рис. 12.2) и выводит их по очереди на экран дисплея.

```
1000 CLS:KEY OFF:WIDTH 40
1010 PRINT"Программа учета личных коллекций – этап просмотра файла"
1030 OPEN "ITEM.DAT" FOR INPUT AS #1
1190 '--Ввод очередной записи файла-----------------------
1200 WHILE NOT EOF(1)
1210 INPUT#1,ITMNR$,CTGRY$,DSCR$(1),DSCR$(2),DSCR$(3)
1220 INPUT#1,IDNBR$,LCN$,ACQRD$,COST#,VALU#
1290 '-- Выдача на экран формы представления информации ----
1300 CLS
1310 PRINT"Программа учета личных коллекций – этап просмотра файла"
1320 RESTORE 7000
1330 FOR ROW%=5 TO 14
1340 READ COL%,LABEL$
1350 LOCATE ROW%,COL%:PRINT LABEL$;STRING$(13-LEN(LABEL$),".")
1360 NEXT
1390 '-- Вывод на экран всех значений ----------------------
1400 LOCATE 5,18:PRINT ITMNR$
1410 LOCATE 6,18:PRINT CTGRY$
1420 LOCATE 7,18:PRINT DSCR$(1)
1430 LOCATE 8,18:PRINT DSCR$(2)
1440 LOCATE 9,18:PRINT DSCR$(3)
1450 LOCATE 10,18:PRINT IDNBR$
1460 LOCATE 11,18:PRINT LCN$
1470 LOCATE 12,18:PRINT ACQRD$
1480 LOCATE 13,18:PRINT USING "######.##";COST#
1490 LOCATE 14,18:PRINT USING "######.##";VALU#
1890 '--Продолжать ввод сообщений или закончить работу?-----
1900 LOCATE 2,1:PRINT "Продолжать просмотр? (Y/N)";TAB(40)
1910 LOCATE 3,1:PRINT TAB(40)'Стереть устаревший текст?
1920 LOCATE 3,1:INPUT;"",NTRY$
1930 IF NTRY$="N" OR NTRY$="n" THEN 2100
1940 WEND
2090 '-- Конец программы ----------------------------------
2100 CLS
2110 PRINT"Программа учета личных коллекций – работа закончена"
2120 PRINT TAB(5);TIME$;TAB(15);DATE$
2130 CLOSE #1
2140 END
6980 '--- Графы и строки ---------------------------------
6990 'Для формы ввода
7000 DATA 4,Номер позиц.,4,Категор.,4,Описание-1,4 , Описание-2,4,
     Описание-3,4 , Идентиф. ,4, Местонах.,4,Дата приобр.,4, Затраты, 4,Стоим.
```

Рис. 12.3. Программа просмотра файла, содержащего сведения о личных коллекциях: выдача информации из файла с последовательным доступом.

Анализ программы создания файла, содержащего сведения о личных коллекциях

Работа этой программы (рис. 12.2) начинается с вопроса к пользователю, желает ли он продолжить занесение записей в уже существующий файл или хочет создать новый (строки 1030 и 1040). В зависимости от ответа пользователя программа открывает последовательный файл ITEM.DAT с режимом доступа FOR OUTPUT или FOR APPEND (строка 1050). Значение переменной D$ — запятая для разделения значений, записываемых в дисковый файл (строка

1060). Для занесения каждой записи в программе используются два оператора PRINT # (строки 1800 и 1810). После того как пользователь закончит ввод позиций, он может либо просмотреть весь файл, либо завершить работу программы (строки 2000—2040).

Для простоты ввод данных с клавиатуры в иллюстративной программе реализован довольно примитивно (строки с 1400 по 1520), однако эту процедуру можно сделать более изящной, используя программу, приведенную на рис. 11.6. Для слияния двух программ вычеркивают (рис. 12.2) строки с 1390 по 1520, с 1900 по 1940, с 2090 по 2130 и то, что осталось, объединяют с текстом (рис. 11.6). Полученная таким образом результирующая программа все же содержит два оператора INPUT (строки 1040 и 2030), которые можно заменить по желанию обращениями к универсальной подпрограмме ввода (как показано в строке 1920 на рис. 11.6).

Анализ программы просмотра файла, содержащего сведения о личных коллекциях

Данная программа (рис. 12.3) поочередно выводит записи на экран дисплея в форме, аналогичной той, что была разработана в гл. 11 (рис. 11.5). Работа программы начинается с открытия файла ITEM.DAT для считывания (строка 1030). Затем до тех пор, пока программа не дойдет до конца файла (строка 1200), она считывает значения из очередной записи (строки 1210 и 1220). Программа выдает на экран определенную форму (строки 1300—1360) и помещает в нее только что считанные значения (строки 1400—1490). После этого формируется запрос, требуется ли считывать следующую запись (строки 1900—1930). При положительном ответе программа продолжает считывание записей, в противном случае ее работа заканчивается.

Файлы с произвольным доступом

В Бэйсике ПВМ имеется ряд операторов, специально предназначенных для чтения и записи информации в файлы с произвольным доступом. Единственные уже знакомые нам операторы такого типа — это OPEN и CLOSE. Есть, однако, и другие операторы, которые используются взамен оператора PRINT # для вывода и INPUT # для ввода. Еще один специальный оператор указывает, какие переменные будут определять значения полей, и даже существуют специальные операторы и функции для присваивания этим переменным новых значений.

Для минимизации числа обращений к диску файлы с произвольным доступом снабжаются буферами. Размер буфера таков, что в нем всегда может поместиться одна запись, а иногда и больше. Когда программе нужна какая-то определенная запись, средства

Бэйсика ПВМ вначале реализуют ее поиск в буфере файла. Если запись удается найти, то обращения к диску не происходит, в противном случае содержимое буфера пересылается обратно в соответствующую область дисковой памяти, а затем уже нужная запись заносится в буфер файла.

Открытие файлов с произвольным доступом

При открытии файла для работы в режиме произвольного доступа ему присваивается номер и устанавливается длина каждой его записи. Все записи одного файла должны быть одинаковой длины.

Пример

```
1010 OPEN "ITEM.DAT" AS #1 LEN=128
```

Отсутствие какого-либо предложения между именем файла и описателем AS в операторе OPEN означает, что файл будет открыт для режима произвольного доступа, при котором разрешаются и чтение, и запись. Такой оператор OPEN должен содержать предложение LEN, указывающее длину записи.

Все записи в файле с произвольным доступом имеют одинаковую фиксированную длину. Важно правильно указать эту длину, чтобы избежать искажения содержимого файла. Обычно максимальная длина записи составляет 128 байт.

Для вычисления необходимой длины записи суммируют длины входящих в нее полей. При этом следует предусмотреть два знака для поля целых чисел, четыре для поля чисел одинарной точности и восемь для поля чисел удвоенной точности. Для поля строковой переменной отводят столько знаков, сколько может потребовать ее возможное значение. Полученные числа заносятся в макет файла (рис. 12.4).

Объявление структуры записи

После открытия файла с произвольным доступом в программе должна быть объявлена структура записей файла. Это реализуется с помощью оператора FIELD.

Пример

```
1020 FIELD #2, 3 AS I$, 10 AS C$, 10 AS D$(1)
```

В операторе FIELD файл, к которому он относится, как обычно, идентифицируется по номеру. Затем перечисляются поля в том порядке, в котором они встречаются в записях файла. Для каждого поля оператор устанавливает длину и имя переменной, которая будет использоваться в программе для идентификации данного поля. Все поля файла должны быть перечислены в одном операторе FIELD и разделены запятыми.

Макет файла

Имя файла ITEM. DAT	Длина записи 126	Число записей 10

Описание
Файл, содержащий сведения о личных коллекциях,— произвольный доступ

Имя переменной	Назначение полей	Максимальная длина	Примечания
ITMNR$	Номер позиции	2	ITMNR%
CTGRY$	Категория предметов	10	
DSCR$(1)	Описание — строка 1	20	
DSCR$(2)	Описание — строка 2	20	
DSCR$(3)	Описание — строка 3	20	
IDNBR$	Идентификационный номер	15	
LCN$	Местонахождение	15	
ACQRD$	Дата приобретения	8	
COST$	Исходная цена (затраты)	8	COST#
VALU$	Стоимость	8	VALU#

Рис. 12.4. Макет файла с произвольным доступом, содержащего сведения о личных коллекциях.

◆◆◆Общая длина всех объявляемых в операторе FIELD полей не должна превосходить длины записи, установленной оператором OPEN. Нарушение этого правила приводит к ошибке.

Все значения пересылаются в файл или из файла с произвольным доступом при помощи переменных, указанных в операторе FIELD; все эти переменные должны быть строковыми. Оператор FIELD удобно задает фиксированную длину каждой указанной в нем переменной. При этом числовые величины должны преобразовываться в строковые с использованием специальных функций, как описано ниже.

Использование переменных оператора FIELD

Присваивание значений переменным, определенным в операторе FIELD, реализуется двумя специальными операторами LSET и RSET.

Пример

```
5340 RSET DAT$=NTRY$
5550 LSET LCN$=BLDG$+ROOM$
```

◆◆◆Ни одну из переменных оператора FIELD нельзя использовать ни в каких модификациях оператора INPUT, и ни одной из них нельзя присваивать значение в операторе LET (с командным словом LET или без него). Нарушение любого из этих правил приводит к утрате соответствия, установленного оператором FIELD, вследствие чего программа не может больше использовать такую переменную для занесения нужных значений в записи файла с произвольным доступом или для их извлечения.

Операторы LSET и RSET гарантируют, что длина значения переменной будет приведена в соответствие с длиной, указанной в операторе FIELD. Если присваиваемое значение содержит слишком мало знаков, оно дополняется пробелами. Оператор LSET выравнивает это значение по левым разрядам, дополняя пробелы справа, а оператор RSET наоборот. Если присваиваемое значение имеет слишком большую длину, то оба оператора отсекают символы справа.

Пример

```
list
10 OPEN "SAMPLE.DAT" AS #1 LEN=20
20 FIELD #1, 10 AS A$
30 RSET A$ = "DISK"
40 PRINT A$
run
      DISK
Ok
```

Числовые величины в дисковом файле

Для хранения в файле с произвольным доступом числовых величин последние необходимо преобразовать в цифровые строки. Для выполнения указанной операции в Бэйсике ПВМ имеются три функции: MKI$, MKS$ и MKD$, которые переводят числовые величины соответственно в строковые переменные в виде целых чисел и чисел с обычной и двойной точностью. Результатом преобразования всегда является строка цифр, содержащая соответственно два, четыре и восемь символов. В случае преобразования больших числовых значений применение указанных функций дает значительно большее сокращение длины строки по сравнению с другими преобразующими функциями.

Функции MKI$, MKS$ и MKD$ не переводят числовые величины в эквивалентные им знаки кода ASCII, как это делается при использовании других функций. Они просто упаковывают соответствующие значения в два, четыре или восемь знаков по той же схеме, которая реализуется в ПВМ для представления числовых величин в динамической памяти. Именно поэтому с помощью оператора PRINT невозможно выдать на печать или экран дисплея результаты работы функций MKI$, MKS$ и MKD$.

Преобразованные числовые величины должны быть затем присвоены переменным файла с помощью операторов LSET или RSET.

Пример

```
4700 LSET COST$=MKD$(NTRY#)
```

Численные значения строковых переменных, полученные с помощью функций MKI$, MKS$ и MKD$, могут быть снова преобразованы в числовые величины соответственно функциями CVI, CVS и CVD.

Запись в файлы с произвольным доступом

Когда программа готова к занесению целиком всей записи в файл с произвольным доступом, для выполнения этой операции должен использоваться оператор PUT#, подобный следующему:

```
1550 PUT #1,ITMNR%
```

Первая величина в операторе PUT# указывает номер файла, а вторая — номер записи; последний указывать не обязательно, и если номер записи опущен, то средства Бэйсика ПВМ присваивают ей номер на единицу больше последнего использованного в данном файле.

Считывание записей из файлов с произвольным доступом

Оператор GET# реализует считывание из файла с произвольным доступом сразу всей записи целиком. Далее для получения значений конкретных полей в программе могут использоваться имена переменных, перечисленные в операторе FIELD.

Пример

```
4020 GET#1, ITMNR%
```

Первая величина в операторе GET# — номер файла, а вторая — номер записи; если он отсутствует, то автоматически используется номер записи, на единицу больший, чем последний номер записи, использованной с тем же номером файла.

Признак конца файла

Программы могут производить считывание или запись за пределами обычного признака конца файла с произвольным доступом, и это не будет приводить к прямым ошибкам. Если в операторе GET# указан номер записи, больший любого использонанного ранее в том же файле оператором PUT#, то в Бэйсике ПВМ каждому символу переменной любого поля присваивается кодовый номер, равный нулю. (Функции CVI, CVS и CVD преобразуют такой код

в число 0.) Оператор PUT#, в котором указан номер записи, больший, чем любой другой в файле, увеличивает размер файла настолько, чтобы можно было поместить указанную запись. Любые не используемые записи между новой и прежней записью, имеющей самый большой номер, будут содержать ненужную информацию. Поэтому оператор GET#, отыскивающий одну из таких промежуточных записей, произведет считывание «мусора», который в дальнейшем может явиться причиной ошибки где-нибудь в другом месте программы. Во избежание таких косвенных ошибок в программах должны накладываться ограничения на номера записей, к которым разрешается обращение.

Файл с произвольным доступом не имеет ограничений на длину, за исключением ограничения по объему свободной дисковой памяти. Функция LOF указывает, сколько места в памяти занимает конкретный файл — с точностью до ближайших 128 байт. С помощью функции LOC определяется текущий номер записи, т. е. номер последней записи, использованной в операторах GET# или PUT# применительно к данному файлу.

Пример

```
1650 IF LOC(1)=150 THEN CLOSE:END
```

Величина, стоящая в круглых скобках, указывает номер файла.

Сигнализация об ошибках

Использование дисковых файлов данных делает программы чувствительными к различным ошибкам, вызывающим преждевременные остановы. Например, когда дискет заполняется до конца, появляется сообщение «Disk full...» («Диск заполнен...»), файлы автоматически закрываются и программа прекращает работу. Другой пример — ошибочная установка защитной этикетки против записи на тот дискет, на который программа все же должна записывать данные. В этом случае появляется сообщение "Disk Write Protected..." («Диск защищен от записи...») и программа прекращает работу.

Бэйсик-программа способна управлять обработкой более 70 типов ошибок (не все они, конечно, связаны с использованием дисков), в противном случае эти ошибки могли бы приводить к срыву выполнения программы. Возможность не прерывать работу программ обеспечивается оператором ON ERROR GOTO; оператор указывает номер строки, к которой должен осуществляться переход, если будет обнаружена ошибка.

Пример

```
ON ERROR GOTO 30000
```

Большинство программистов помещают подобный оператор в первую часть программы и располагают специальную стандартную программу обработки встречающихся ошибок в строке с указанным номером. Действие этой программы определяется характером ошибки. Если, например, произошла ошибка, связанная с блокировкой по защите диска от записи, программа обработки ошибок может предоставить пользователю выбор: закончить выполнение программы или зафиксировать сбойную ситуацию и повторить операцию.

Программа обработки ошибок может проверять значение функции ERR для определения характера возникшей ошибки. Ошибки идентифицируются кодовыми номерами (перечень кодов ошибок и их описание приведены в Приложении С). Иногда одного только номера ошибки бывает недостаточно для однозначной ее идентификации. Например, в случае открытия программой более одного файла есть риск, что при выполнении каждого оператора OPEN произойдет ошибка, связанная с защитой от записи, и может оказаться необходимым обрабатывать каждую такую ошибку по-разному. Функция ERL сообщает поэтому номер строки, в которой произошла последняя ошибка, что может быть использовано программой для дифференцирования ошибок, имеющих одинаковый код.

Программа обработки ошибок должна заканчиваться оператором RESUME. Если в какой-то программе первым выполняется оператор STOP, END или RETURN, это тоже приводит к ошибке. Оператор RESUME имеет три формы. Простейшей формой является отдельное командное слово; оно завершает программу обработки ошибок возвратом к тому оператору, при выполнении которого была обнаружена ошибка. Вторая форма — это оператор RESUME NEXT, обеспечивающий обратный переход к оператору, следующему за тем, при выполнении которого была обнаружена ошибка. Третья разновидность позволяет указать номер строки, к которой следует перейти.

Пример

```
RESUME 30100
```

Пользователь, безусловно, не захочет писать индивидуальную программу для обработки каждого из более чем 70 типов ошибок. Поэтому в Бэйсике ПВМ предусмотрена возможность передачи управления стандартной подпрограмме с помощью оператора ON ERROR GOTO 0. В этом случае работа программы прекращается, и выдается стандартное сообщение об ошибке.

Бэйсик ПВМ обеспечивает одновременную обработку только одной ошибки. Если оператор ON ERROR GOTO отсылает какую-

```
10 CHAIN MERGE "11-4",20 'Слияние с универсальной подпрограммой ввода
20 CHAIN MERGE "11-7",1000 'и изменениями к ней
890 '--Программа обработки ошибок ----------------
900 IF ERR=61 AND ERL=1120 THEN 950
910 ON ERROR GOTO 0 'Обработка произвольных ошибок
920 STOP: GOTO 910
940 'Диск заполнен; повторное открытие файла
950 OPEN "ITEM.DAT" AS #1 LEN=126
960 FILSIZ%=FIX(LOF(1)/126)
970 PUT #1,FILSIZ%
980 RESUME 1140
990 '++Начало основной программы++++++++++++++++
1000 CLS:KEY OFF:WIDTH 40
1010 PRINT"Программа учета личных коллекций-изменение файла"
1020 FILSIZ%=10'Задание макс. числа записей для нового файла
1030 RETKEY$=CHR$(17)+CHR$(196)+CHR$(217) 'Образ клавиши <--'
1040 OPEN "ITEM.DAT" AS #1 LEN=126
1050 FIELD #1, 2 AS ITMNR$, 10 AS CTGRY$, 20 AS DSCR$(1), 20 AS
    DSCR$(2), 20 AS DSCR$(3), 15 AS IDNBR$, 15 AS LCN$, 8 AS ACQRD$,
    8 AS COST$, 8 AS VALU$
1060 IF LOF(1)>0 THEN 1200
1070 '-- Инициализация пустого файла ------------------------
1080 PRINT"Пожалуйста, приготовьтесь к инициализации файла"
1090 ON ERROR GOTO 900 Выявление ошибки из-за переполнения диска
1100 LSET ITMNR$=MKI$(-1) Инициализ. всех записей для позиции с номером-1
1110 WHILE LOC(1)<FILSIZ%
1120 PUT #1
1130 LOCATE 5,10:PRINT"Позиция";LOC(1);"инициализирована";
1140 WEND
1150 '-Пауза для чтения пользователем информации на экране --
1160 LOCATE 7,1:PRINT "Максимально допустимый номер позиции"
1170 LOCATE 9,6,1:PRINT "Для продолжения нажмите клавишу пробела";
1180 K$=INKEY$:IF K$<>" " THEN 1180
1190 '-- Задание номера записи, подлежащей изменению или вводу
1200 CLS
1210 PRINT"Программа учета личных коллекций-изменение файла"
1220 LOCATE 2,1:PRINT "Введите номер позиции (или END, если это конец)"
1230 LOCATE 3,4:PRINT "  (Для след. позиции нажмите только';RETKEY$;".)
1240 INLEN%=3:INMAX#=FILSIZ%:INMIN#=0:INROW%=3:INCOL%=1: INTMPL$=
    "*":GOSUB 200
1250 IF NTRY$="end" OR NTRY$="END" THEN 2100 'Конец программы
```

Рис. 12.5. Программа учета личных коллекций, работающая с файлами с произвольным доступом.

либо программу к подпрограмме обработки ошибок, в которой возникает еще одна ошибка, эта ошибка не будет обработана, но она вызовет появление сообщения об ошибке и останов.

Использование файлов с произвольным доступом

Описанные в данной главе две программы учета личных коллекций можно улучшить путем использования файлов с произвольным доступом, что позволит незамедлительно обновлять такие быстроменяющиеся данные, как местоположение и стоимость. Макет, при-

```
1260 IF NTRY#>0 THEN ITMNR%=NTRY#:GOTO 1280
1270 IF NTRY#=0 AND LOC(1)<FILSIZ% THEN ITMNR%=LOC(1)+1 ELSE BEEP:
     GOTO 1240
1280 GET #1,ITMNR% 'Поиск нужной записи
1290 '-- Выдача на экран формы ввода ----------
1300 RESTORE 7000
1310 FOR ROW%=5 TO 14
1320 READ COL%,LABEL$
1330 LOCATE ROW%,COL%:PRINT LABEL$;STRING$(13-LEN(LABEL$),".")
1340 NEXT
1370 '--Ввод знач. (для новых записей) или показ тек. знач. (для существующих записей)
1380 IF CVI(ITMNR$)>0 THEN 1460 ' Сравнение с пробелом?
1390 '-- Последовательный ввод всех записей ----------
1400 LOCATE 5,22:PRINT "(Новая позиция)"
1410 FOR N%=1 TO 10
1420 GOSUB 4000
1430 NEXT
1440 GOTO 1600 ' Обход повторной выдачи значений на экран
1450 '--Вывод на экран текущих значений ----------
1460 LOCATE 5,18:PRIN   NI(ITMNR$)
1470 LOCATE 6,18:PRINT CTGRY$
1480 LOCATE 7,18:PRINT DSCR$(1)
1490 LOCATE 8,18:PRINT DSCR$(2)
1500 LOCATE 9,18:PRINT DSCR$(3)
1510 LOCATE 10,18:PRINT IDNBR$
1520 LOCATE 11,18:PRINT LCN$
1530 LOCATE 12,18:PRINT ACQRD$
1540 LOCATE 13,18:PRINT USING "######.##";CVD(COST$)
1550 LOCATE 14,18:PRINT USING "######.##";CVD(VALU$)
1590 '-- Разрешение изменений входных сообщений ----------
1600 N%=2 ' Начальная установка указателя против позиции 2
1610 LOCATE 2,1:PRINT"Для ввода отмеченного сообщ. нажмите клавишу"; CHR$(26);"
1620 PRINT CHR$(24);" и  ";CHR$(25);"передвиньте указатель. Оконч. считыв. клав ."
1630 LOCATE N%+4,2,0:PRINT CHR$(16); ' Высвечивание указателя
1640 K$=INKEY$:IF LEN(K$)=0 THEN 1640 ' Ожидание нажатия клавиши
1650 PRINT CHR$(29);" "; ' Гашение указателя
1660 '-- Игнорирование бессмысленных нажатий клавиш
1670 IF LEN(K$)<>2 THEN BEEP:GOTO 1630 ELSE K%=ASC(RIGHT$(K$,1))
1680 IF K%=77 THEN GOSUB 4000:GOTO 1610 ' Стрелка вправо
1690 IF K%=79 THEN 1800 ' Отпускание клавиши
1700 IF K%=72 AND N%>2 THEN N%=N%-1:GOTO 1630 ' Стрелка вверх
```

Рис. 12.5. (Продолжение.)

веденный на рис. 12.4, показывает структуру подобного файла, а программа, представленная на рис. 12.5, предназначена для создания новых записей, просмотра существующих и внесения в записи необходимых изменений.

Анализ усовершенствованной программы учета личных коллекций

Для работы улучшенной программы учета личных коллекций (рис. 12.5) необходима универсальная подпрограмма ввода (см. рис. 11.4). Слияние обеих программ и добавление изменений (рис. 11.7)

```
1710 IF K%=80 AND N%<10 THEN N%=N%+1:GOTO 1630 ' Стрелка вниз
1720 BEEP:GOTO 1630 ' Бессмысленное нажатие клавиши
1790 '-- Занесение одиночной записи в файл данных ----------
1800 PUT#1,ITMNR%
1810 GOTO 1200
2090 '-- Конец программы ----------
2100 CLS
2110 PRINT"Программа учёта личных коллекций-Конец";TAB(39);
2120 PRINT TAB(5);TIME$;TAB(15);DATE$
2130 CLOSE #1
2140 END
3990 '== Ввод сообщения с номером N% ==========
4000 LOCATE 2,1,0:PRINT" Вводите по шаблону      ";TAB(40)
4010 LOCATE 3,1:PRINT Нажмите;RETKEY$; для ввода сообщений (40)
4020 INMAX#=10+20:INMIN#=-INMAX#:INROW%=N%+4:INCOL%=18:INTMPL$="*"
4050 ON N% GOTO 4100,4200,4300,4300,4300,4400,4500,4600,4700,4800
4090 '-- Номер позиции ----------
4100 LOCATE INROW%,INCOL%:PRINT ITMNR%
4110 LSET ITMNR$=MKI$(ITMNR%):RETURN
4120 PRINT SPACE$(INLEN%-LEN(NTRY$));:RETURN
4190 '-- Категория ----------
4200 INLEN%=10:GOSUB 200:LSET CTGRY$=NTRY$:GOTO 4120
4290 '-- Описание  1,2, & 3----------
4300 INLEN%=20:GOSUB 200:LSET DSCR$(N%-2)=NTRY$:GOTO 4120
4390 '-- Идентификатор ----------
4400 INLEN%=15:GOSUB 200:LSET IDNBR$=NTRY$:GOTO 4120
4490 '-- Местонахождение ----------
4500 INLEN%=15:GOSUB 200:LSET LCN$=NTRY$:GOTO 4120
4590 '-- Дата приобретения ----------
4600 INLEN%=8:GOSUB 200:LSET ACQRD$=NTRY$:GOTO 4120
4690 '-- Затраты ----------
4700 INMAX#=999999.99#:INMIN#=0:INLEN%=9:GOSUB 200:LSET COST$=MKD$(NTRY#)
4710 '-- Воспроизведение введенных чисел на экране ----------
4720 LOCATE INROW%,INCOL%:PRINT USING "######.##"; NTRY#;:RETURN
4790 '-- Стоимость ----------
4800 INMAX#=999999.99#:INMIN#=0:INLEN%=9:GOSUB 200:LSET VALU$=MKD$(NTRY#)
4810 GOTO 4720
6980 '--- Графы и строки ----------
6990 ' for entry form
7000 DATA 4,Ном. позиции.,4,Категория,4, Описание-1,4,  Описание-2,4,
     Описание-3,4,  Идентиф.,4,Местонах.,4,Дата приобрет,4,Затраты, 4, Стоим.
```

Рис. 12.5. (Продолжение.)

осуществляются с помощью операторов CHAIN MERGE (строки 10 и 20 на рис. 12.5).

Работа основной программы начинается с задания режима работы экрана (строки 1000 и 1010) и присвоения значений паре переменных (строки 1020 и 1030). Переменная FILSIZ% устанавливает максимальный номер записи в файле ITEM.DAT. Переменная RETKEY$ содержит символы, необходимые для воспроизведения на экране изображения, соответствующего клавише ↵.

Затем программа открывает файл ITEM.DAT, содержащий информацию о коллекциях, для произвольного доступа и определяет переменные полей (строки с 1040 по 1050). Если файл пуст, то про-

грамма инициализирует ввод каждой записи, присваивая соответствующей позиции номер —1 (строки с 1060 по 1140). В случае переполнения диска в процессе ввода записей фиксируется ошибка (строка 1090). Программа обработки ошибок (строки с 900 по 960) снова открывает файл и блокирует дальнейшую инициализацию. По окончании инициализации всех записей программа останавливается, чтобы дать возможность пользователю считать с экрана дисплея самый последний задействованный номер позиции (строки с 1160 по 1180).

Когда файл, содержащий информацию о коллекциях, готов, программа запрашивает у пользователя первый номер позиции (строки 1200—1240) и отыскивает соответствующую запись в файле (строки 1260—1280). Если пользователь нажимает клавишу ↵, программа автоматически находит следующую запись (строки 1270 и 1280). При вводе пользователем слова "END" или "end" работа программы завершается (строка 1250).

Если данная запись никогда прежде не использовалась, она будет иметь номер позиции —1 (благодаря установленной процедуре инициализации новых файлов в соответствии со строками 1060—1140). В этом случае пользователь должен ввести все необходимые значения переменных до того как у него будут запрошены изменения (строки 1400—1440). В противном случае программа выводит на экран значения, считанные ею из дискового файла (строки 1460—1550).

При выводе на экран дисплея текущих значений величин в записи программа разрешает пользователю изменять любое значение, кроме номера позиции (строки 1610—1720), во избежание нарушения целостности файла. Пользователь выбирает определенное значение и вносит соответствующие изменения с помощью клавиш управления курсором (гл. 11; рис. 11.9 и 11.10). Когда пользователь заканчивает внесение изменений в запись, программа заносит ее в файл и совершает возврат на прежнюю ветвь для взятия очередного номера позиции.

Отметим, что в рассмотренной программе переменные полей нигде не вводятся с помощью оператора INPUT и им нигде не присваиваются значения с использованием оператора LET.

Изменение ограничений для дисковых файлов

Обычно средства Бэйсика ПВМ налагают ограничения на число одновременно открываемых файлов и устройств (три) и на максимальную длину записи файла с произвольным доступом (128 знаков). Эти ограничения можно изменить, когда ДОС ПВМ передает управление вычислительным процессом интерпретатору дискового или расширенного Бэйсика.

Чтобы изменить максимальное число одновременно открываемых

файлов и устройств, нужно добавить справа в команду BASIC или BASICA операционной системы ДОС ПВМ пробел, символы "/F:" и указать желаемое число открытых файлов и устройств.

Пример. Команда

```
A>basica /f:5
```

инициирует работу со средствами расширенного Бэйсика и разрешает одновременное открытие пяти файлов и устройств (максимально возможное их число — 15).

Обычно размер буфера определяется максимальной длиной записи файла с произвольным доступом. Это соглашение можно изменить, добавив справа в команду BASIC или BASICA операционной системы ДОС ПВМ пробел, символы "/S:" и величину желаемого размера буфера.

Пример. По команде

```
B>a:basic /s:512
```

запускаются средства дискового Бэйсика, который теперь будет работать с буфером объемом 512 байт (символов). Максимально допустимый размер буфера файла — 32767 байт [1].

Можно реализовать и совместное действие сразу обоих вариантов изменения ограничений, описанных выше, добавив их в любом порядке в конец команды BASIC или BASICA, например так:

```
A>basic /f:5 /s:512
```

Использование дополнительно открываемых файлов или буферов увеличенной емкости связано с перерасходом машинных ресурсов, поскольку для каждой позиции требуется 190 байт динамической памяти плюс еще 128 байт под буфер файла, если только не реализованы описанные выше изменения. Таким образом, увеличение числа одновременно открываемых файлов или расширение буфера файла (либо то и другое вместе) приводит к уменьшению полезного объема динамической памяти для размещения операторов и переменных программы.

[1] Для получения наилучших результатов при работе с накопителями на дискетах, выпускаемыми фирмой IBM, рекомендуется использовать буфера емкостью 512 байт.

Глава 13

ГРАФИЧЕСКИЕ СРЕДСТВА

Используя любую из трех предназначенных для ПВМ версий Бэйсика, можно легко перейти от текстового вывода данных к их графическому выводу. Кассетный, а также дисковый и расширенный Бэйсик включают в себя средства вывода на экран отдельных точек, построения линий, прямоугольников. Последний, кроме того, располагает некоторыми дополнительными возможностями графического вывода: он позволяет проводить окружности, дуги, эллипсы, заполнять любую заданную область экрана однотонным цветом. Единственным требованием к оборудованию для реализации графического вывода является наличие цветного графического адаптера и цветного экрана дисплея любого типа. Именно вычислительные системы, в состав которых входит указанное оборудование, являются предметом рассмотрения данной главы.

Передний план, фон и окантовка

На экране дисплея можно выделить три отдельные области: *передний план*, *фон* и *окантовку*.

Передний план — область, где располагаются текстовые данные и графические изображения, которые накладываются на фон.

Фон, как это следует из самого названия,— область экрана, в которой воспроизводится все, что выводится на экран; фон можно видеть сразу после включения ПВМ.

Окантовка окружает фон, но, как правило, она бывает того же цвета, что и сам фон, и потому неразличима. Благодаря наличию окантовки сглаживаются различия между экранами разнотипных телевизионных установок и мониторов.

Обычно в телевизионных устройствах на экране воспроизводится только часть принимаемого сигнала, а какая-то его часть оказывается за кадром. Подобное явление называется *разверткой за пределами рабочей области экрана*. В одних бытовых телевизорах оказывается невидимой почти вся окантовка, в других — только некоторая ее часть, в мониторах же потери еще менее существенны. Чтобы изображение умещалось в поле экрана дисплея, окантовку можно сдвинуть вправо или влево с помощью команды ДОС ПВМ MODE.

Режимы работы дисплея при текстовом и графическом выводе

Тексты и специальные символы из числа 256 стандартных знаков, допустимых для ПВМ (Приложение D), выводятся на экран дисплея с помощью операторов PRINT и PRINT USING. Соответ-

ствующий режим работы дисплея называется *текстовым*. Наличие цветного графического адаптера позволяет использовать дисплей еще в двух режимах графического вывода (применительно к Бэйсику). Работая в этих режимах, можно выводить на экран отдельные точки, вычерчивать на нем линии и прямоугольники, окружности и дуги, а также закрашивать внутренние области очерченных контуров.

Графические режимы при работе с Бэйсиком ПВМ отличаются один от другого числом и размером выводимых точек, а также количеством допустимых цветов. В режиме *высокого разрешения* экран разбивается на большее число точек, чем в режиме *среднего разрешения*. Вследствие этого в первом случае точки оказываются вдвое меньшего размера, благодаря чему изображение может содержать существенно большее число деталей. В режиме высокого разрешения изображение всегда черно-белое; при работе же со средней разрешающей способностью возможно появление на экране одновременно до четырех различных цветов.

Переключение режимов работы экрана

Переход от режима текстового вывода к любому режиму графического вывода и наоборот осуществляется с помощью оператора SCREEN. Например, переключение в режим графического вывода со средней разрешающей способностью осуществляется оператором

```
200 SCREEN 1
```

Значение, стоящее после командного слова, идентифицирует конкретный режим работы экрана: 1 определяет режим графического вывода со средней разрешающей способностью, 2 — работу с высоким разрешением, а 0 — режим текстового вывода. В соответствии с этим в результате выполнения приводимого ниже оператора происходит обратный переход в режим текстового вывода:

```
800 SCREEN 0
```

Если указанный в операторе SCREEN режим совпадает с текущим, никаких действий не производится. В противном случае при выполнении SCREEN происходит очистка экрана дисплея, устанавливается белый цвет для переднего плана и черный для фона.

Оператор COLOR в текстовом режиме. Данный оператор позволяет задавать различные цвета для выводимого на переднем плане текста, фона и окантовки. Точный оттенок каждого цвета меняется в зависимости от используемого монитора — его марки, модели и настройки.

Цвета идентифицируются номерами (табл. 13.1), например

```
210 COLOR 5,1,7
```

Таблица 13.1. **Цвета для текстового режима**

Номер цвета [1])	Цвет	Номер цвета [2])	Цвет
0	Черный	8	Темно-серый
1	Синий	9	Светло-синий
2	Зеленый	10	Светло-зеленый
3	Голубой	11	Светло-голубой
4	Красный	12	Розовый
5	Пурпурный	13	Светло-пурпурный
6	Золотистый (коричневый)	14	Желтый
7	Белый (серый)	15	Ярко-белый

[1]) Если к номеру цвета переднего плана прибавить 16, то выводимые символы будут мерцать.
[2]) При задании цвета фона номера с 8 по 15 определяют те же цвета, что и номера с 0 по 7.

Первое стоящее после командного слова значение определяет цвет для переднего плана, второе — для фона, а третье — для окантовки. В приведенном выше примере для переднего плана задан пурпурный цвет, для фона — синий, а для окантовки — белый.

Номера цветов от 0 до 7 допустимы как для переднего плана и окантовки, так и для фона, тогда как номера с 8 по 15, соответствующие более светлым оттенкам основных цветов 0—7, разрешается использовать только для переднего плана и окантовки. Некоторые цветные мониторы генерируют один и тот же цвет для номеров 8 и 0, 9 и 1, 10 и 2 и т. д., несмотря на то что от вычислительной машины поступают разные сигналы.

В текстовом режиме оператор COLOR оказывает влияние на цвет переднего плана и фона только для тех символов, которые будут выводиться *после* его выполнения. Текст, уже высвеченный на экране к моменту выполнения такого оператора COLOR, не изменит своего цвета, а для всех последующих появляющихся на экране символов цвет переднего плана и фона будет новым. В отличие от этого цвет окантовки изменяется сразу.

Пример. Использование оператора COLOR в режиме немедленной обработки.

```
color  11,5,14
print  "Графический дисплей"
Графический дисплей
Ok
```

В данном примере оператор COLOR изменяет цвет переднего плана на светло-голубой, фона — на пурпурный, а окантовки — на желтый. Сам оператор COLOR высвечивается на экране с преж-

ним цветом переднего плана и фона, а оператор PRINT и выведенные в результате его выполнения данные появляются на экране в светло-голубом цвете на пурпурном фоне.

Любое из трех чисел, задаваемых в операторе COLOR, можно опустить, и тогда цвет соответствующей области экрана не изменится.

Пример. При выполнении оператора

```
320 COLOR ,3
```

цвет фона изменится на голубой, а цвет окантовки и переднего плана останется прежним.

Если в текстовом режиме в операторе COLOR перед первым значением стоит запятая (как в последнем примере), то при выполнении оператора это значение будет интерпретироваться как номер нового цвета фона, а цвет переднего плана не изменится. Оператор COLOR с двумя запятыми перед первым значением вызывает изменение только цвета окантовки.

Пример. При выполнении оператора

```
8733 COLOR ,,2
```

цвет окантовки становится зеленым, а цвет переднего плана и фона остается прежним.

В текстовом режиме на самой нижней строке экрана высвечиваются определения 10 функциональных клавиш (гл. 10), причем эти определения всегда выводятся черным цветом на белом фоне. С помощью оператора KEY OFF их можно удалить с экрана и сделать цвет 25-й строки таким же, как и у остальных.

Оператор COLOR в графическом режиме. Данный оператор как бы объединяет в себе три различных оператора: в текстовом режиме его действие одно, в графическом режиме со средней разрешающей способностью — другое, а в графическом режиме с высоким разрешением, когда цвет переднего плана всегда белый, а фона и окантовки черный, этот оператор вообще недопустим.

В режиме графического вывода со средней разрешающей способностью номер цвета фона определяет также и цвет окантовки. При этом разрешается использовать любой из 16 цветов, допустимых для переднего плана в текстовом режиме вывода. Номер цвета фона определяет, кроме того, оттенок цвета переднего плана — светлый или темный. Номера цветов от 0 до 7 соответствуют темной окраске фона и переднего плана, а от 8 до 15 — светлым тонам этих областей. Номера же цветов от 16 до 23 определяют темный оттенок для фона и светлый для переднего плана. Соответствие между номерами, цветами и оттенками описано в табл. 13.2.

Возможности задания цвета переднего плана в режиме со средней разрешающей способностью довольно ограниченны, а способ задания конкретного цвета в этом случае существенно отличается

Таблица 13.2. **Цвета фона в графическом режиме со средней разрешающей способностью**

Цвет	Номер цвета фона [1]		
	Темный передний план, темный фон	Светлый передний план, светлый фон	Светлый передний план, темный фон
Черный	0	8	16
Синий	1	9	17
Зеленый	2	10	18
Голубой	3	11	19
Красный	4	12	20
Пурпурный	5	13	21
Золотистый	6	14	22
Белый	7	15	23

[1]) Номер цвета фона определяет его цвет и оттенок, а также оттенок переднего плана.

от того, как это делается в текстовом режиме. В режиме среднего разрешения имеются два набора цветов для переднего плана, называемые *палитрами*. Каждая палитра состоит из четырех цветов. Оператор COLOR определяет только, какая из двух палитр будет активной, а конкретный цвет из четырех имеющихся задается в самих операторах, выполняющих графические построения. Как уже указывалось выше, оттенок цвета переднего плана задается цветом фона.

Палитры, что вполне естественно, идентифицируются номерами 0 и 1, а цвета каждой из них — номерами от 0 до 3 (табл. 13.3). Заметим, что для каждой палитры цвета с номерами 1, 2 и 3 фиксированы, а 0 определяет цвет, совпадающий с цветом фона. Цвет текста,

Таблица 13.3. **Цвета переднего плана в графическом режиме вывода со средней разрешающей способностью [1])**

Номер палитры	Номер цвета палитры			
	0	1	2	3
0	Цвет фона	Зеленый	Красный	Золотистый
1	Цвет фона	Голубой	Пурпурный	Белый

[1]) Палитра цветов переднего плана выбирается с помощью оператора COLOR; конкретные цвета выбранной палитры задаются в самих операторах, выполняющих графические построения. Оттенок цвета переднего плана определяется номером цвета фона (табл. 13.2).

выводимого на экран в графическом режиме со средней разрешающей способностью, обычно соответствует коду 3, т. е. золотистый или белый.

Пример. Использование оператора COLOR в режиме графического вывода со средним разрешением.

```
1000 COLOR 1,0
```

Первое числовое значение определяет цвет фона и границы, а второе — палитру цветов переднего плана (в текстовом режиме задаваемые в операторе COLOR значения имеют прямо противоположную интерпретацию). Любое из этих двух значений, но не сразу оба, можно опустить, и тогда цвет соответствующей области (или областей) не изменится.

При выполнении оператора COLOR в графическом режиме со средним разрешением сразу же изменяется цвет всего фона и всего переднего плана, включая и то, что было выведено на экран ранее. Например, если на экран были выведены рисунки, окрашенные в цвета палитры 0, т. е. в зеленый, красный или золотистый, то переключение на палитру 1 немедленно вызовет изменение этих цветов на голубой, пурпурный и белый.

Координаты точек экрана в графических режимах

Чтобы выводить точки, строить линии, прямоугольники и т. д., нужно иметь возможность задавать то место на экране, где они должны располагаться. Для этого необходимо уметь однозначно идентифицировать любую точку экрана. Представим себе лист бумаги, разграфленный в клетку и наложенный на экран так, что каждая клетка соответствует отдельной точке экрана. Теперь предположим, что строки и столбцы разграфленной бумаги пронумерованы. Тогда любую клетку можно идентифицировать соответствующими ей номером строки и номером столбца, которые называются ее *координатами*; именно в этом и состоит способ задания местоположения точек на экране графического дисплея.

Столбцы нумеруются, начиная с крайнего левого, которому присваивается номер 0, и кончая крайним правым, которому соответствует номер 319 в графическом режиме со средним разрешением или номер 639 в режиме высокого разрешения. И в том и в другом случае самая верхняя строка имеет номер 0, а самая нижняя — 199. Прежде чем использовать строки со 192-й по 199-ю, необходимо убедиться в том, что высвеченные в нижней части экрана определения функциональных клавиш предварительно были удалены с экрана с помощью оператора KEY OFF.

Оператор PSET

Ни оператор COLOR, ни SCREEN сами по себе не выполняют на экране никаких графических построений; они только подготавливают экран к работе в графическом режиме со средним или высоким разрешением.

Оператор PSET предназначен для отображения на экране, работающем в одном из графических режимов, отдельной точки с заданными координатами; этот оператор имеет один и тот же вид в обоих допустимых режимах. Например, при выполнении оператора

```
1010 PSET (45,10)
```

на экране появится точка на пересечении 45-го столбца и 10-й строки.

Номер цвета в операторе PSET задавать не обязательно. Если этого номера нет, то в графическом режиме со средним разрешением выводится золотистая или белая точка (цвет, соответствующий номеру 3), а в режиме с высокой разрешающей способностью — белая.

Для того чтобы задать в операторе PSET цвет выводимой точки, после скобок с координатами следует поставить запятую и номер требуемого цвета. В режиме со средней разрешающей способностью для задания одного из четырех цветов активной палитры используются номера от 0 до 3 (табл. 13.3). В режиме с высокой разрешающей способностью нечетные числа определяют единственный допустимый в этом режиме цвет переднего плана (белый), а четные — единственный цвет фона (черный).

Пример

```
1020 PSET (180,150), 1
```

При выполнении данной команды в режиме среднего разрешения на экране появится зеленая или голубая точка — в зависимости от того, какая палитра является активной; в режиме с высокой разрешающей способностью на экран будет выведена белая (цвета переднего плана) точка.

Оператор LINE

Во все версии Бэйсика ПВМ входит достаточно мощное средство организации графического вывода — оператор LINE. С его помощью можно чертить на экране отрезки прямых линий, контуры прямоугольников, закрашивать одним цветом внутренние области четырехугольников.

Поскольку любая прямая линия определяется двумя точками, вполне естественно, что в операторе LINE для выводимого отрезка прямой должны быть заданы две точки — начальная и конечная. Для этого можно в явном виде указать координаты обеих точек.

Пример

```
1030 LINE (50,150)—(1,10)
```

При выполнении данного оператора на экране будет построен отрезок прямой, начало которого лежит на пересечении 50-го столбца и 150-й строки, а конец — в точке пересечения 1-го столбца и 10-й строки.

В операторе LINE можно также задавать и цвет выводимой линии. Для этого достаточно после координат конечной точки записать номер требуемого цвета. В графическом режиме со средней разрешающей способностью такой номер определяет один из четырех цветов активной палитры (табл. 13.3); в режиме с высоким разрешением нечетные номера соответствуют цвету переднего плана, а четные — цвету фона. Если номер цвета в операторе LINE не задан, как в приведенном выше примере, то в режиме среднего разрешения используется цвет с номером 3, а в режиме с высокой разрешающей способностью — цвет переднего плана.

Построение ломаных линий

С помощью нескольких операторов LINE можно вычертить на экране несколько отрезков прямой так, чтобы каждый последующий отрезок начинался в той точке, где закончился предыдущий.

Пример

```
1000 SCREEN 1 'Средняя разрешающая способность
1010 COLOR 0,0 'Черный фон, палитра 0
1040 LINE (60,140)—(100,90),1
1050 LINE (100,90)—(140,140),2
1060 LINE (140,140)—(100,190),1
1070 LINE (100,190)—(60,140),2
```

При выполнении четырех приведенных выше операторов LINE на экране строится ромб, противоположные стороны которого одинакового цвета, а смежные — разного.

Для выполнения подобных построений существует более удобный способ, который основан на том, что всякий раз при выводе на экран какой-либо линии в памяти ПВМ запоминаются координаты последней выведенной точки, которыми можно воспользоваться при выполнении очередного оператора LINE. Если в операторе LINE опустить координаты начальной точки отрезка, то вычерчиваемый с помощью такого оператора отрезок будет начинаться в последней участвующей в предыдущих построениях точке и кончаться в точке, заданной в данном операторе.

Пример. Четыре оператора LINE, строящие тот же ромб, что и четыре оператора LINE предыдущего примера.

```
1000 SCREEN 1, 'Средняя разрешающая способность
1010 COLOR 0,0 'Черный фон, палитра 0
1040 LINE (60,140)—(100,90),1
1050 LINE —(140,140),2
1060 LINE —(100,190),1
1070 LINE —(60,140),2
```

В первом из приведенных операторов LINE начальная точка задана, поскольку координаты последней выведенной перед этим на экран точки неизвестны. Начальную точку можно было бы также установить, отобразив ее с помощью оператора PSET. В любом случае во всех последующих операторах LINE достаточно задать только конечную точку.

Вообще говоря, если на экране предварительно не производилось никаких графических построений, то по умолчанию предполагается, что начальная точка совпадает с центром экрана, т. е. в качестве начальной будет использоваться точка с координатами (160,100) в режиме среднего разрешения или с координатами (320, 100) в режиме с высокой разрешающей способностью.

Построение прямоугольников

С помощью оператора LINE специального вида на экране можно построить прямоугольник. Все, что нужно для этого сделать,— дописать в конце обычного оператора LINE после номера цвета запятую и букву B.

Пример

```
1100 LINE (50,70)—(70,90),2,B
```

При выполнении данного оператора в режиме со средней разрешающей способностью на экране появится красный или ярко-пурпурный квадрат в зависимости от того, какая палитра является активной. Одна из вершин этого квадрата будет располагаться в точке с координатами (50,70), а диагонально противоположная ей вершина — в точке с координатами (70, 90). Поскольку в качестве номера цвета в операторе указано четное число, в режиме с высокой разрешающей способностью цвет квадрата будет совпадать с цветом фона, т. е. квадрат будет невидимым.

Если буква B в операторе LINE присутствует, а номер цвета не задан, то по умолчанию будет использоваться цвет с кодом 3. Заметим, что, хотя сам номер цвета может быть опущен, запятая, которая стояла бы после него, должна оставаться на месте, указывая на неявное задание цвета.

Пример. Оператор LINE, выполняющий построение узкого вытянутого вверх прямоугольника, окрашенного в цвет с номером 3.

```
1110 LINE (150,10)—(155,190),,B
```

Оператор LINE с буквой B в конце позволяет вычерчивать на экране только квадраты и прямоугольники, стороны которых параллельны сторонам экрана. Чтобы построить четырехугольник произвольной формы, необходимо в общем случае выполнить четыре отдельных оператора LINE.

Закрашивание прямоугольника одним цветом

С помощью оператора LINE можно построить не только контуры прямоугольника, но и прямоугольник, закрашенный одним цветом. Для этого достаточно добавить в оператор LINE непосредственно после буквы B букву F.

Пример

```
1120 LINE (0,195)—(319,199),1,BF
```

При выполнении данного оператора на экране появится длинная узкая сплошная полоса, проходящая вдоль нижнего края экрана и закрашенная в цвет с номером 1.

Относительные координаты

Существуют два способа идентификации точки, отображаемой на экране графического дисплея. Во всех приведенных до сих пор примерах точка всегда задавалась соответствующими ей номерами столбца и строки. Такой способ задания точки называется *абсолютной координатной идентификацией.* Альтернативой такому способу является так называемая *относительная координатная идентификация,* при которой координаты задаются относительно последней выведенной на экран точки. Если в операторах PSET или LINE собственно координатам точки предшествует слово "STEP", то эти координаты интерпретируются как смещение относительно координат последней выведенной на экран точки.

Пример

```
1200 PSET (40,23)
1210 PSET STEP (10,—3)
```

В первом из приведенных операторов (строка 1200) для вывода точки на пересечении 40-го столбца и 23-й строки используются абсолютные координаты, а во втором операторе (строка 1210) — относительные. В последнем случае на экран будет выведена точка на пересечении 50-го столбца и 20-й строки, т. е. на 10 столбцов правее и на 3 строки выше предыдущей.

В операторах графического вывода точку можно идентифицировать любым способом — как с помощью абсолютных координат, так и с помощью относительных. Если относительными координатами задается самая первая выводимая точка, то интерпретатор будет использовать в качестве предыдущей точки центр экрана.

Смешанное использование текста и графики

С помощью операторов PRINT и PRINT USING можно выводить любые из 256 допустимых для ПВМ символов во всех режимах работы экрана. Обычно выводимые символы окрашены в цвет с номером 3, т. е. в режиме со средней разрешающей способностью они белые или золотистые в зависимости от того, какая палитра активна, и в режиме с высокой разрешающей способностью белые. При средней разрешающей способности выводимые символы имеют такую же величину, как и в текстовом режиме при 40-символьной строке экрана. В режиме высокого разрешения символы вдвое уже, т. е. такие же, как в случае текстового вывода при 80-символьной строке.

Местоположение символов на экране задается в операторах LOCATE, PRINT и PRINT USING (и только в них) одинаково во всех трех режимах работы экрана. Операторы графического вывода, подобные PSET и LINE, на размещение символов никакого влияния не оказывают (гл. 10).

Координаты текстового режима, задаваемые в операторе LOCATE, по существу аналогичны координатам, используемым в графическом режиме, хотя некоторые различия все же имеются. Так, при текстовом режиме строки и столбцы нумеруются, начиная с 1, а не с 0, как при графическом. Кроме того, в операторе LOCATE номер строки должен стоять перед номером столбца, а в координатах, задаваемых операторами графического режима, наоборот.

При выводе символов предполагается, что экран содержит всегда 25 строк с номерами от 1 (для самой верхней строки) до 25 (для нижней строки) и 40 (средняя разрешающая способность) либо 80 столбцов (высокая разрешающая способность). Таким образом, одна строка текстового режима соответствует восьми строкам графического, а символ в ширину занимает восемь графических столбцов.

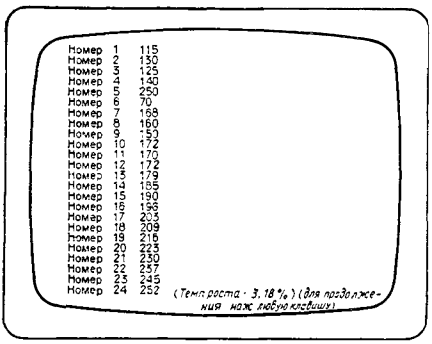
Если текст накладывается на уже выполненные графические построения, то элементы изображения будут частично стираться: вокруг каждого символа стирается окружающая его прямоугольная область шириной 8 столбцов и высотой 8 строк. В то же время точки, отрезки прямых линий и прямоугольники, вычерчиваемые поверх уже выведенных символов, могут лишь немного изменить контуры тех символов, с которыми они пересекаются.

В режиме немедленной обработки при выводе на экран команд поверх уже существующих графических построений неожиданно может появиться сообщение "Syntax error" («Синтаксическая ошибка»). Это может произойти потому, что в текущую заполняемую экранную строку могут случайно попадать фрагменты графических построений, окрашенные в тот же цвет, что и текст выводимой на экран команды, и тогда интерпретатор Бэйсика ПВМ будет пытать-

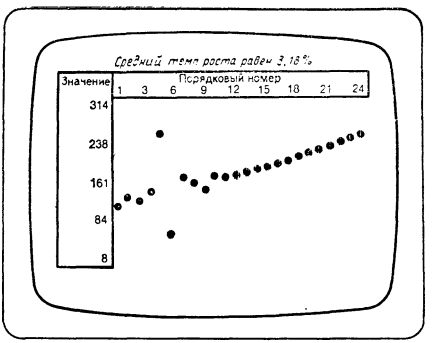
ся интерпретировать их как обычные символы. Во избежание таких ошибок перед вводом с клавиатуры команд в режиме немедленной обработки следует всегда предварительно очистить весь экран или по крайней мере отдельную его строку (с помощью клавиши **Esc**).

Использование графических средств построения точек, линий и прямоугольников

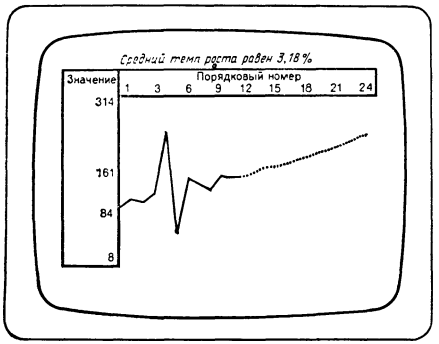
Многие программы, в результате работы которых выдается лишь некоторый список значений (рис. 13.1.A), можно сделать более удобными для использования, если включить в них средства графического вывода. Вычерчиваемые графики могут состоять из отдельных точек (рис. 13.1B), представлять собой ломаные линии (рис. 13.1C) или гистограммы (рис. 13.1D). График можно строить с использованием нескольких цветов, делая его тем самым более наглядным и одновременно более информативным.



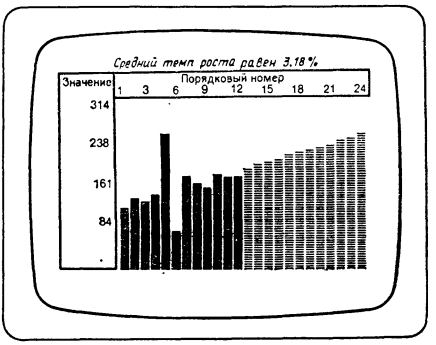
A *Выведенный на экран список числовых значений*



B *Точечный график*



C *Кусочно-линейный график*



D *Гистограмма*

Рис. 13.1. Текстовый и графический вывод результатов выполнения программы.

Программа построения графика средних темпов роста

Программа определения средних темпов роста (рис. 13.2) вычисляет средний темп роста по заданному набору показателей и выдает прогноз изменения этих показателей в будущем. По результатам прогнозирования формируется список точных значений, после этого — точечный график, затем — обычный график в виде лома-

```
50 DIM AMT(24)
1010 CLS:KEY OFF:WIDTH 40
1100 PRINT
1110 PRINT "В данной программе используется"
1120 PRINT "для анализа прошедшего этапа и прогнозов"
1130 PRINT "метод экспоненциальной регрессии:"
1140 PRINT "Вам необходимо задать значения прошедшего"
1150 PRINT "этапа и общее количество всех значений"
1160 PRINT "в прошлом и будущем, всего не более 24."
1170 PRINT
1200 INPUT "Сколько значений в прошлом?";PAST%
1210 INPUT" Сколько значений нужно предсказать? ";FUTR%
1220 IF PAST%+FUTR%>24 THEN PRINT:PRINT "Не используйте больше 24
     значений":GOTO 1170
1230 PRINT
1240 PRINT "Введите значения прошедшего этапа:"
1250 PRINT
1260 FOR NBR%=1 TO PAST%
1270 PRINT" Значение";NBR%;
1280 INPUT AMT(NBR%)
1290 'Вычисление вспомогат. величин для использ. метода экспоненц. регрессии
1300 X=NBR%-1:Y=LOG(AMT(NBR%))
1310 I1=I1+X:I2=I2+Y
1320 I3=I3+X^2:I4=I4+Y^2
1330 I5=I5+X*Y
1340 NEXT NBR%
1390 '--Вычисление коэффициентов экспоненциальной регрессии ---
1400 B=(PAST%*I5-I2*I1)/(PAST%*I3-I1^2)
1410 A=(I2-B*I1)/PAST%
1420 RATE=CINT((EXP(B)-1)*10000!)/100
1490 '--Прогнозирование будущих значений ----------
1500 FOR NBR%=PAST%+1 TO PAST%+FUTR%
1510 AMT(NBR%)=INT(EXP(A)*EXP(B*(NBR%-1))+.5)
1520 NEXT NBR%
1590 '--Вывод на экран точных числовых значений ----------
1600 CLS
1610 FOR NBR%=1 TO PAST%+FUTR%
1630 PRINT
1640 PRINT"Значение";NBR%;TAB(11);AMT(NBR%);
1650 NEXT NBR%
```

Рис. 13.2. Программа определения средних темпов роста.

ной линии и, наконец, гистограмма. В программе используется метод статистической обработки, называемый *методом экспоненциальной регрессии*, который позволяет предсказывать будущее поведение значений, экспоненциально увеличивающихся или уменьшающихся во времени. Типичными приложениями этого метода являются проблема оценки рождаемости и похожие задачи прогнозирования сбыта, доходов и клиентуры.

```
1660 PRINT TAB(18);"(Темп роста:";RATE;"%)";
1690 LOCATE 25,6:PRINT"Для продолж.нажмите любую клавишу";:A$=
     INPUT$(1)
1990 '--Вычисление масштабн. коэфф.для вертикальной шкалы---
2000 MIN=AMT(1)
2010 MAX=AMT(1)
2020 ' Поиск максимального и минимального значений
2030 FOR NBR%=1 TO PAST%+FUTR%
2040 IF AMT(NBR%)>MAX THEN MAX=AMT(NBR%)
2050 IF AMT(NBR%)<MIN THEN MIN=AMT(NBR%)
2060 NEXT NBR%
2070 'Выбор масштаба, при котором в отведенной области помест. весь график
2080 SCALE=168/(MAX+MIN)
2190 '--- Разметка экрана для вывода графика ----------------
2200 CLS:SCREEN 1'Установка графич. режима со средн. разреш. способностью
2210 COLOR 23,0 'Lt.Светло-серый (белый) фон, палитра: зел./красный/желтый
2220 PRINT TAB(6);" Средний темп роста равён";RATE;"%"
2230 PRINT "Amount";TAB(22);"Номер"
2240 PRINT TAB(8);"1  3   6   9  12  15  18  21  24";
2250 ' Вывод на экран вертикальной шкалы
2260 FOR CTR%=0 TO 4
2270 LOCATE 24-CTR%*5,1 'В строках 4, 9, 14, 19, & 24
2280 PRINT USING "###### ";(40*CTR%+4)/SCALE;' Вывод значения
2290 NEXT CTR%
2300 ' Окаймление шкал
2310 LINE (51,7)-(319,23),3,B
2320 LINE (0,7)-(51,191),3,B
2390 '---Постр.точечного графика ----------------------------
2400 FOR NBR%=1 TO PAST%+FUTR%
2410 IF NBR%<FUTR% THEN HUE=2 ELSE HUE=1 'Выбор цвета выводимой точки
2420 PSET (NBR%*11+45,191-CINT(SCALE*AMT(NBR%))),HUE
2430 NEXT NBR%
2490 '--- Постр. кусочно-лин.графика----------------------------
2500 A$=INPUT$(1)' Ожидание нажатия клавиши
2510 HUE=2 ' Цвет для значений прошедшего этапа
2520 PSET (56,191-CINT(SCALE*AMT(1))),HUE
2530 FOR NBR%=2 TO PAST%+FUTR%
2540 IF NBR%>PAST% THEN HUE=1 ' Изменение цвета для прогноза
2550 LINE -(NBR%*11+45,191-CINT(SCALE*AMT(NBR%))),HUE
```

Рис. 13.2. (Продолжение.)

```
2560 NEXT NBR%
2590 '--Построение гистограммы----------------------------
2600 A$=INPUT$(1)'Ожидание нажатия клавиши
2610 HUE=2'Цвет для значений прошедшего этапа
2620 LINE (52,24)-(319,191),0,BF'Очистка области экрана, отведенной под график
2630 FOR NBR%=1 TO PAST%+FUTR%
2640 IF NBR%>PAST% THEN HUE=1 'Изменение цвета для прогноза
2650 LINE (NBR%*11+45,191)-(NBR%*11+54,191-CINT(SCALE*AMT(NBR%))),HUE,BF
2660 NEXT NBR%
2690 '--Конец программы-----------------------------------
2700 A$=INPUT$(1)'Ожидание нажатия клавиши
2710 SCREEN 0
2720 END
```

Рис. 13.2. (Продолжение.)

В начальной части программы определяется массив числовых значений с обычной точностью (строка 50) и производится подготовка экрана дисплея к последующему вводу данных с клавиатуры (строки 1010—1250).

Для краткости и простоты в программе используются самые элементарные средства ввода с клавиатуры без применения более сложной техники ввода, описанной в гл. 11. Пользователь программы должен разбить весь анализируемый период на два этапа — прошедший и будущий (строки 1200 и 1210), при этом общее количество значений, относящихся к прошлому и будущему, не должно превышать 24 (строка 1220). Затем пользователь должен ввести известные значения, т. е. числа, соответствующие прошедшему этапу (строки с 1260 по 1340).

По мере ввода данных в программе происходит накопление вспомогательных числовых величин, которые потребуются в дальнейшем для применения метода экспоненциальной регрессии (строки 1300—1340). Сразу же после окончания стадии ввода исходных данных выполняются заключительные этапы метода экспоненциальной регрессии — вычисление коэффициентов экспоненциальных уравнений (строки 1400—1420). Затем в программе решается задача прогнозирования: подсчитываются значения, относящиеся к будущему этапу (строки с 1500 по 1520).

При выполнении операторов строк 1600—1690 на экран выводится полный набор численных значений, соответствующих прошедшему и будущему этапам. После этого программа переходит в состояние ожидания и находится в нем до тех пор, пока пользователь не ознакомится со всей выведенной на экран информацией и не нажмет какую-нибудь клавишу для возобновления выполнения программы (строка 1690).

Поскольку нет каких-либо ограничений на диапазон изменения вводимых и прогнозируемых значений, в программе должен определяться нужный масштаб графиков, с тем чтобы они полностью

умещались на экране (это делается в строках программы с 2000 по 2080). При масштабировании сначала определяются наибольшее и наименьшее значения (строки 2000—2060), а затем максимально допустимая высота графика, равная 168 строкам, делится на сумму наибольшего и наименьшего значений (строка 2080). В результате такого деления получается масштабный коэффициент вертикальной шкалы.

На этапе подготовки к построению графиков в программе устанавливается графический режим вывода со средним разрешением, выбираются цвет фона и палитра цветов переднего плана (строки 2200 и 2210). Затем на экран выводится информация, относящаяся к заголовку и координатным осям графика (строки с 2220 по 2320). Полезно обратить внимание на то, как используются операторы LINE для построения прямоугольников, обрамляющих горизонтальную и вертикальную шкалы (строки 2310 и 2320).

Если горизонтальная шкала выводится на экран с помощью одного простого оператора PRINT (строка 2240), то для вывода вертикальной шкалы приходится выполнять более сложные действия. Прежде всего в программе вычисляются пять опорных точек, перекрывающих весь диапазон известных и вычисленных значений; эти точки получаются путем деления номеров строк 4, 44, 84, 124 и 164 на выбранный масштабный коэффициент по вертикали (программные строки 2260—2290). Перечисленные строки вертикальной шкалы соответствуют серединам обычных экранных строк текстового режима, имеющих номера 4, 9, 14, 19 и 24; именно в эти строки экрана помещаются полученные опорные значения. (Отметим для ясности, что график занимает на экране 168 строк, с 24-й по 191-ю.)

Точечный график строится в программе с помощью оператора PSET (строки 2400—2430). Переменная HUE обеспечивает изменение цвета выводимых точек, с тем чтобы все известные заранее точки делались красными, а все прогнозируемые — зелеными. Для каждой точки ее координата по столбцам вычисляется путем умножения номера этой точки, изменяющегося от 1 до 24, на коэффициент растяжения горизонтальной шкалы, равный 11, и прибавления к полученному произведению 45, т. е. номера крайнего левого столбца отведенной под график области. Для вычисления координаты по строкам масштабный коэффициент вертикальной шкалы умножается на известное или прогнозируемое значение, и из полученного произведения вычитается 191 — номер самой нижней строки занимаемой графиком области.

После того как пользователь нажмет какую-нибудь клавишу, точечный график сменяется ломаной линией (строки 2500—2560). Первая точка строится с помощью оператора PSET (строка 2520), а отрезки прямой, соединяющие остальные точки, проводятся с помощью оператора LINE (строка 2550). Координаты по строкам и столбцам вычисляются здесь, как и в предыдущем случае, и, так же

как и ранее, переменная HUE определяет цвет выводимой точки.

В заключение на экран выводится гистограмма (строки 2600—2660). На этом этапе сначала производится очистка занимаемой графиком области экрана путем вывода на экран прямоугольника, закрашенного в цвет фона (строка 2620). Затем для каждого значения, соответствующего прошлому или будущему, строится вытянутый вверх прямоугольник, закрашенный одним цветом (строки с 2630 по 2660). По ширине все эти прямоугольники одинаковы, а высота каждого из них определяется величиной соответствующего значения. Цвет прямоугольника задается переменной HUE.

Построение окружностей и закрашивание замкнутых областей

Бэйсик позволяет легко оперировать прямыми линиями, поэтому обычно почти все выполняемые машиной графические построения сводятся к выводу отдельных точек, вычерчиванию линий и построению прямоугольников. Однако в расширенный Бэйсик дополнительно входит специальный оператор CIRCLE, позволяющий точно так же легко и быстро строить окружности, дуги, эллипсы.

Расширенный Бэйсик располагает еще одним оператором графического вывода PAINT, который особенно удобно использовать вместе с оператором CIRCLE. Оператор PAINT предназначен для заполнения одним цветом заданной области экрана и тем самым позволяет рисовать на экране закрашенный круг так же легко, как и чертить окружность.

Оператор CIRCLE

Для того чтобы с помощью оператора расширенного Бэйсика CIRCLE построить на экране окружность, необходимо указать, в каком месте экрана она должна располагаться, какого она должна быть размера и цвета. Кроме того, поскольку CIRCLE является оператором общего назначения, позволяющим строить как целые окружности, так и отдельные их части, необходимо также задать, какая именно часть окружности должна изображаться на экране.

Пример. Оператор, выполняющий построение полной окружности в середине экрана, работающего в режиме графического вывода со средней разрешающей способностью.

```
CIRCLE (160,100),50,1,0,6.2831
```

Первые два числа определяют координаты центра окружности; в данном случае центр будет находиться на пересечении 160-го столбца и 100-й строки. Третье число, равное здесь 50, задает размер окружности. Четвертое число указывает, каким цветом должна изображаться окружность, а последние два числа определяют, какая часть окружности должна быть построена.

Координаты центра окружности задаются обычным способом, т. е. с помощью стандартных номеров столбцов и строк. Размер окружности определяется ее радиусом, причем единицей измерения радиуса является ширина одного столбца экрана. Таким образом, окружность с радиусом, равным 6, занимает по диаметру 12 столбцов. Вообще говоря, ширина одного столбца не совпадает с шириной строки, т. е. клетка экрана не является квадратной — в ширину она больше, чем в высоту. Однако при выполнении оператора CIRCLE это различие может оказаться незаметным, поскольку в режиме со средней разрешающей способностью шесть столбцов в ширину занимают столько же места, сколько пять строк, а в режиме высокого разрешения 12 столбцов по ширине соответствуют пяти строкам. Так, если в режиме со средней разрешающей способностью диаметр окружности равен 12 столбцам, то он одновременно равен 10 строкам.

При вычерчивании окружности в режиме среднего разрешения можно использовать любой цвет активной палитры; при этом оттенок выбранного цвета будет определяться номером текущего цвета фона (табл. 13.2 и 13.3). В режиме с высокой разрешающей способностью нечетные номера определяют белый цвет окружности, а четные — цвет, совпадающий с цветом фона. Номер цвета, равный 0, всегда определяет цвет, совпадающий с цветом фона; этот номер удобно использовать для стирания с экрана заданной области.

Пример

```
10 SCREEN 1 'режим графического вывода со средней разрешающей способностью
20 CLS
30 CIRCLE (200,120),25,1,0,6.2831 'голубая
40 CIRCLE (110,140),25,3,0,6.2831 'белая
50 A$=INPUT$(1) 'Ожидание нажатия клавиши
60 CIRCLE (110,140),25,0,0,6.2831 'цвет фона
```

При выполнении этой программы сначала на экране будут построены две окружности, одна голубая, другая белая, а затем программа перейдет в состояние ожидания и будет находиться в нем до тех пор, пока пользователь не нажмет какую-нибудь клавишу. Тогда программа сотрет белую окружность путем вывода на экран точно такой же окружности, но не белого цвета, а цвета фона. Задавать цвет в операторе CIRCLE не обязательно.

Пример. Оператор CIRCLE в режиме высокого разрешения с опущенным номером цвета.

```
10 CIRCLE (320,100),100, ,0,6.2831
```

◆◆◆ Заметим, что можно опускать номер цвета, но не следующую за ним запятую. При выполнении оператора CIRCLE без указания цвета на экран выводится окружность стандартного (принятого по

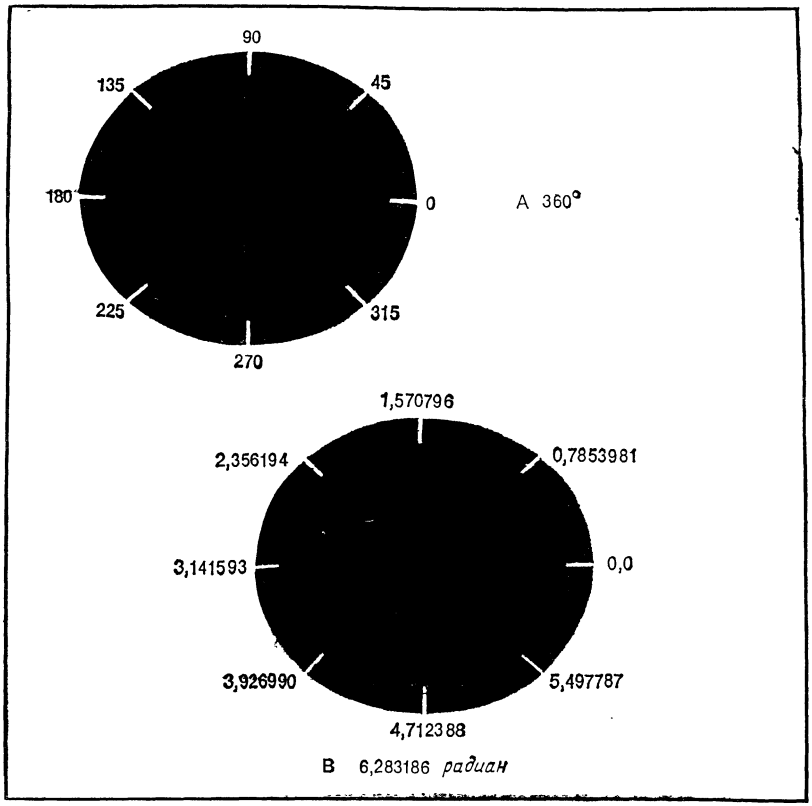


Рис. 13.3. Измерение дуг окружностей (360° равны 6;283186 рад).

умолчанию) цвета переднего плана. В режиме со средней разрешающей способностью стандартным является цвет с номером 3 (белый или золотистый), а в режиме с высокой разрешающей способностью — цвет с номером 1 (белый).

Построение дуг окружности. С помощью оператора CIRCLE можно строить отдельные части окружности, но для этого необходим некоторый способ задания произвольной части окружности. Как известно из геометрии, любая полная окружность содержит 360°. Именно поэтому, например, шкала компаса разделена на 360 равных частей. Однако нумерация градусов в компасе не такая, как принято в геометрии. Поясним принятый в геометрии способ нумерации градусов, используя циферблат часов. Нумерация начинается с 0°, что соответствует положению часовой стрелки на 3-часовой отметке, и продолжается против часовой стрелки так, что 90° соответствуют 12 ч, 180° — 9 ч, 270° — 6 ч и, наконец, 360° (как и 0°) соответствуют 3 ч (рис. 13.3. А).

Дальнейшее осложенние связано с тем, что в расширенном Бэйсике дуги окружности измеряются не в градусах, а в *радианах* (рис. 13.3. В). Понятие *радиан* связано с математической константой π, которая приближенно равна 3,14159. Половина окружности, т. е. дуга в 180°, содержит π радиан, а полная окружность, составляющая 360°, 2π радиан, что приближенно равно 6,2831. Таким образом, для перехода от градусов к радианам достаточно умножить выраженное в градусах значение на 0,0174532.

Чтобы идентифицировать в операторе CIRCLE дугу, необходимо задать точку на окружности, в которой эта дуга начинается, и точку, где она кончается. Обе точки должны задаваться в радианах. Во всех приведенных до сих пор примерах оператора CIRCLE использовалась дуга, начинающаяся при 0 и кончающаяся при 6.2831 радианах, т. е. полная окружность.

Пример. Оператор, в котором задается ровно половина окружности.

```
10 CIRCLE (160,100),50,1,0,3.1416
```

В указанном способе задания дуг остается некоторая неоднозначность — две точки, задающие дугу, в действительности определяют две дуги: одна дуга получается, если от начальной точки двигаться к конечной против часовой стрелки, а другая — если двигаться по часовой стрелке. Для устранения этой неоднозначности в расширенном Бэйсике принято следующее соглашение:

если значение, определяющее начальную точку, меньше значения для конечной точки, то дуга проводится против часовой стрелки, а если первое значение больше второго, то — по часовой стрелке (рис. 13.4).

Построение радиусов. Кроме дуг с помощью оператора CIRCLE можно изображать на экране радиус, соединяющий центр заданной дуги с одной из ее концевых точек. Если концевая точка дуги задается в операторе CIRCLE отрицательным значением, то при выполнении такого оператора будет проведен радиус в эту концевую точку. Знак минус при задании концевой точки не оказывает никакого влияния на саму дугу.

Пример. Оператор, при выполнении которого на экране появится радиус, соединяющий точку с координатами (40,100) со второй концевой точкой дуги.

```
1230 CIRCLE (40,100),30, ,3.1416,—4.7124
```

Значение —0 не интерпретируется так, как любое другое отрицательное значение, используемое при задании концевой точки. Для того чтобы обойти это ограничение, можно вместо —0 указывать —0.001.

Если обе концевые точки заданы отрицательными значениями, то проводятся оба радиуса, образуя сектор.

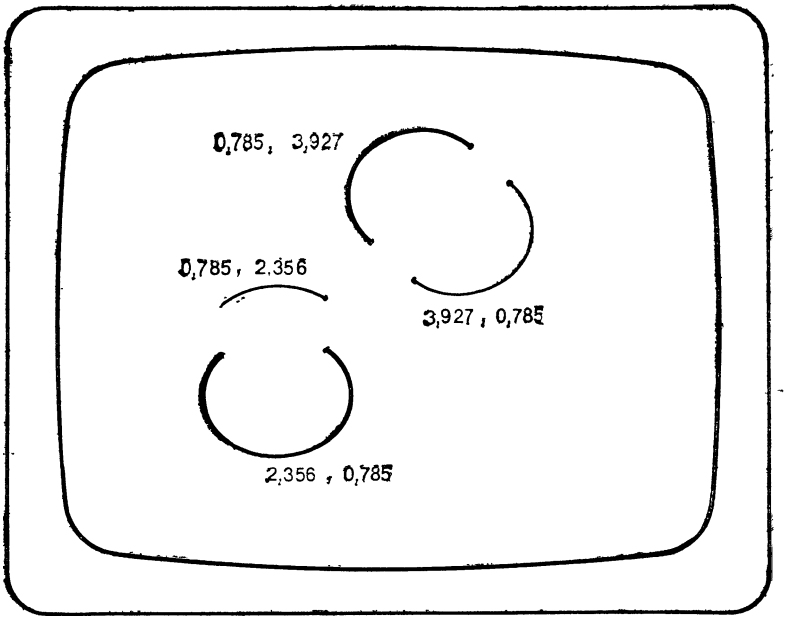


Рис. 13.4. Примеры дуг, которые определяются своими концевыми точками.

Пример

```
1350 CIRCLE (240,100),30,2,—3.1416,—4.7124
```

Построение эллипсов. С помощью оператора CIRCLE можно также изображать на экране эллипсы. Для этого в операторе необходимо задать *характеристическое отношение* изображаемого эллипса, которое является отношением высоты эллипса к его ширине, например

```
1410 CIRCLE (270,100),50,1,0,6.2831,5/3
```

Дополнительное дробное значение в конце оператора CIRCLE определяет характеристическое отношение. Удобнее и проще всего рассматривать характеристическое отношение эллипса как некоторую дробь с заданными числителем и знаменателем. Числитель указывает, какое количество строк при выполнении оператора CIRCLE следует считать эквивалентным числу столбцов, задаваемому знаменателем. В режиме со средней разрешающей способностью характеристическое отношение 5/6 определяет окружность; отношение 1/3 (или 2/6) — эллипс, вытянутый по горизонтали; отношение 5/3 (или 10/6) — эллипс, вытянутый по вертикали (рис. 13.5). В режиме с высокой разрешающей способностью характеристическое отношение 5/12 соответствует окружности.

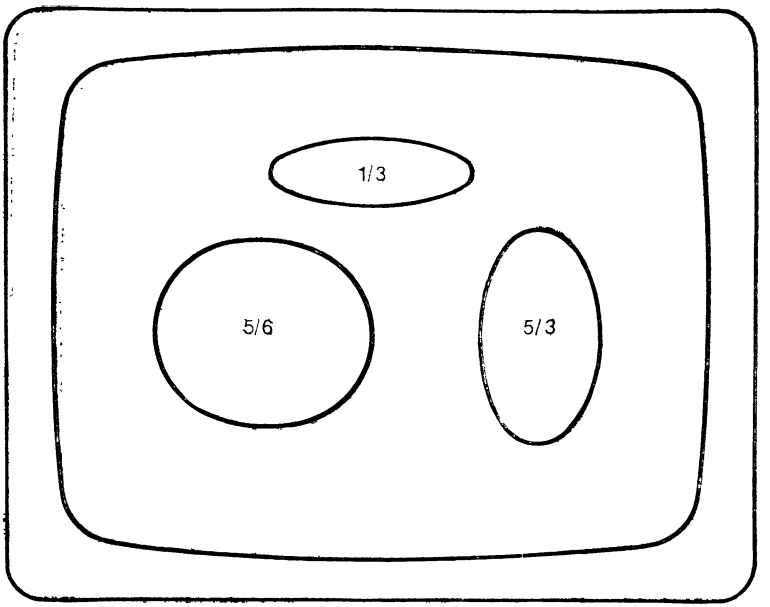


Рис. 13.5. Характеристические отношения эллипсов, изображенных в режиме графического вывода со средней разрешающей способностью.

Характеристические отношения в операторе CIRCLE могут задаваться также и десятичными дробями. Например, отношение 5/6 задает тот же эллипс, что и десятичная дробь .8333333. Интерпретатор расширенного Бэйсика устроен таким образом, что при выполнении операторов CIRCLE с характеристическим отношением, меньшим 1, на экране строятся эллипсы одинаковой ширины и различной высоты, а если характеристическое отношение больше 1, то все соответствующие эллипсы будут иметь одну и ту же высоту и разную ширину (рис. 13.6).

Оператор PAINT

Оператор графического вывода PAINT, входящий в состав расширенного Бэйсика, позволяет заполнять заданную область экрана любым допустимым в режимах графического вывода цветом.

Пример

```
20 PAINT (150,100),1,3
```

При выполнении данного оператора экран начнет закрашиваться цветом с номером 1, начиная с точки (150,100), равномерно по всем направлениям; по каждому направлению это будет продолжаться до тех пор, пока не встретится точка, окрашенная в цвет

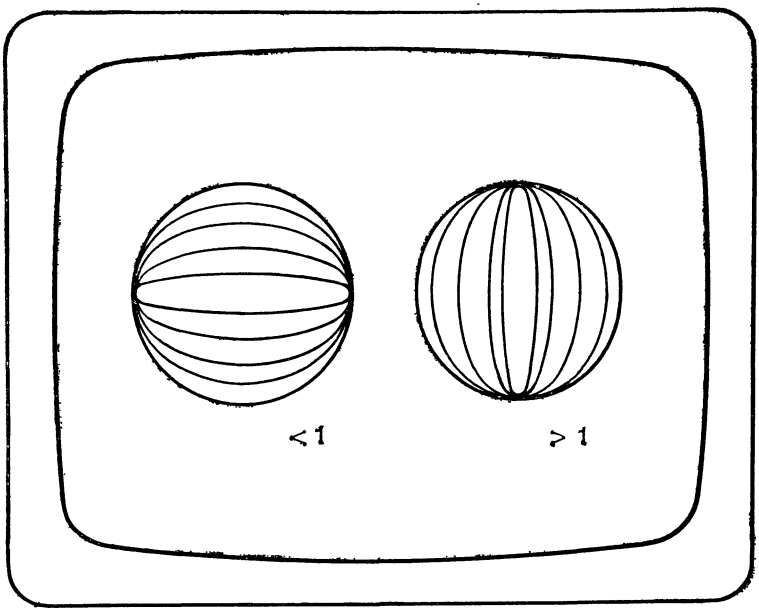


Рис. 13.6. Изменение формы эллипса в зависимости от значения характеристического отношения.

с номером 3. Это значит, что предварительно на экране должна быть проведена замкнутая кривая цвета 3, охватывающая точку (150,100).

Заполняющий цвет и ограничивающий цвет обычно бывают разными, но могут и совпадать. Если нужно закрасить некоторую область, то вся она должна быть ограничена одним и тем же цветом, иначе выполнение оператора PAINT не приведет к ожидаемым результатам. Для вычерчивания границы можно использовать любое сочетание точек, отрезков, прямоугольников, окружностей, дуг и эллипсов. Если линия ограничивающего цвета в некоторых местах прерывается, то заполняющий цвет выходит за пределы области и окрашивает всю оставшуюся часть экрана. Это свойство оператора PAINT дает возможность перекрашивать целиком весь фон в цвет переднего плана.

Пример. Программа, в которой производится перекрашивание фона в цвет переднего плана с номером 1, а затем из новой окраски как бы вырезается круг (т. е. проводится окружность, которая окрашивается в прежний цвет фона).

```
10 SCREEN 1 'режим графического вывода со средней разрешающей способностью
20 CLS
```

```
30 PAINT (0,0),1,1 'закрашивание всего фона
40 CIRCLE (50,150),20,0,0,6.2831 'построение окружности
50 PAINT (50,150),0,0 'заполнение цветом фона
```

Поскольку при выполнении оператора PAINT используется рабочий стек, понятно, что в процессе закрашивания некоторой области на экране может появиться сообщение об ошибке "Out of memory..." («Нехватка памяти...»). Вероятность появления такой ошибки повышается, если оператор PAINT является частью глубоко вложенных друг в друга циклов FOR/NEXT или подпрограмм или же если контур, ограничивающий закрашиваемую область, имеет достаточно сложную форму.

Использование средств построения окружностей и закрашивания замкнутых областей

Существует много примеров практического использования операторов CIRCLE и PAINT. Так, с помощью окружностей удобно изображать пропорциональные соотношения: если некоторому значению, например 20, поставить в соответствие окружность определенного размера, то окружность вдвое большего диаметра будет соответствовать значению 40, а вдвое меньшего — значению 10 (рис. 13.7). Традиционным способом изображения пропорциональных соотношений между частями одного целого, безусловно, является представление частей в виде секторов одной и той же окружности, что также требует использования средств построения окружностей и закрашивания областей (рис. 13.8).

Программа построения пропорциональных окружностей

Такая программа (рис. 13.9) выполняет деление некоторого общего количества на части (допускается использовать до 6 ч.) и выводит на экран окружности, каждая из которых по размеру пропорциональна одной из частей. На рис. 13.8 приводится результат работы программы для исходных значений, равных 20, 40, 3, 7, 10 и 8. В программе используются самые элементарные средства Бэйсика, вследствие чего она может выполняться не для любых наборов исходных значений.

В программных строках 20 и 30 задаются координаты (номера столбцов и строк) для всех шести окружностей, которые могут впоследствии изображаться на экране. Первая окружность будет при этом иметь центр в точке (85,60), вторая — в точке (175,95) и т. д.

В начале программы производится подготовка экрана к вводу данных с клавиатуры (строки 40 и 50). После этого пользователь программы должен ввести значения, соответствующие распределе-

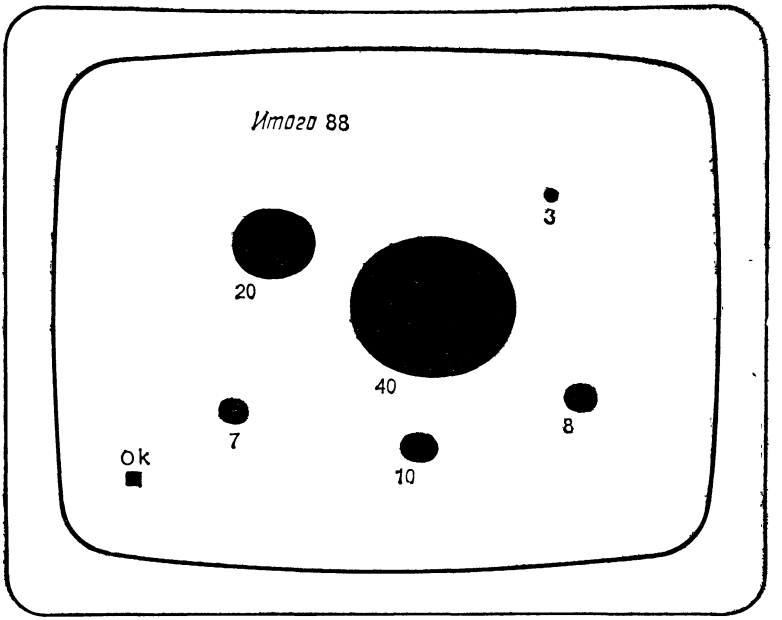


Рис. 13.7. Использование окружностей для изображения пропорциональных соотношений.

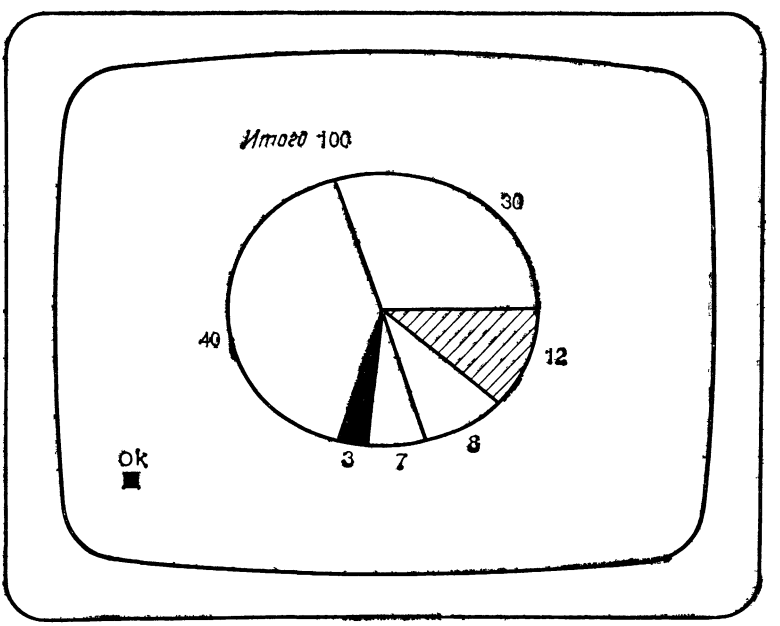


Рис. 13.8. Круговая диаграмма.

```
10 '-- Координаты по столбцам и строкам для центров окружностей --
20 DATA 85,60,175,95,245,35
30 DATA 60,150,165,170,260,145
40 KEY OFF
50 CLS
60 '-- Ввод шести численных значений ----------------------
70 FOR J=1 TO 6
80 PRINT " Значение отдельной части ";J;
90 INPUT PART(J)
100 TOTAL=TOTAL+PART(J)' Подсчет нарастающего итога
110 NEXT J
120 '-- Установка режима работы экрана ---------------------
130 SCREEN 1 'Режим среднего разрешения
140 CLS
150 LOCATE 1,15
160 PRINT TOTAL;"Итого"
170 '--Построение на экране окружностей----------------------
180 FOR J=1 TO 6
190 IF PART(J)=0 THEN GOTO 260 'Игнорир. значения, равного 0
200 READ C,R 'Получение координат центра окружности
210 RAD=100*PART(J)/TOTAL 'Вычисление радиуса окружности
220 LOCATE (R+RAD)/8+2,(C-RAD)/8 ' Опр. позиции для вывода
230 PRINT PART(J); ' Вывод значения
240 CIRCLE (C,R),RAD,2,0,6.2831 'Построение окружности
250 PAINT (C,R),1,2 ' Закрашивание круга
260 NEXT J
270 LOCATE 22
280 END
```

Рис. 13.9. Программа построения пропорциональных окружностей.

нию общего количества по частям (строки с 70 по 110). Среди вводимых могут встречаться значения, равные 0. В процессе ввода в программе постепенно накапливается общая сумма всех значений (строка 100). После завершения ввода программа переводит экран в режим графического вывода со средней разрешающей способностью и выводит на экран заголовок (строки с 130 по 160).

После этого программа переходит к выводу на экран окружностей, пропорциональных по размеру исходным значениям (строки с 180 по 260). Если какое-либо из шести исходных значений равно 0, то оно игнорируется, т. е. окружность для него не строится (строка 190). Для каждого ненулевого значения сначала из списка операторов DATA выбираются координаты точки, которая будет центром соответствующей окружности (строка 200). Затем на основе отношения рассматриваемого исходного значения к общей сумме всех значений вычисляется радиус окружности (строка 210). После этого на экране вблизи будущей окружности выводится исходное

значение (строки 220 и 230), и, наконец, на экране строится окружность (строка 240), которая затем закрашивается (строка 250).

По окончании построения всех окружностей программа перемещает курсор в нижнюю часть экрана (строка 270), подготавливая его к выводу сообщения "Ok", которое появится после завершения работы программы.

Программа построения круговой диаграммы

Данная программа (рис. 13.10) вычерчивает на экране круговую диаграмму, содержащую до 25 секторов. Пользователь программы должен ввести количество исходных значений и сами эти значения. На рис. 13.9 изображены результаты работы программы для шести исходных значений: 30, 40, 3, 7, 8 и 12. В результате работы программы на экране изображается круговая диаграмма, в которой около дуги каждого сектора проставлено соответствующее этому сектору значение.

В начальной части программы отводится память под массив, состоящий из 25 элементов — для ввода до 25 исходных значений (строка 10). Затем выполняются действия по подготовке экрана к вводу данных с клавиатуры в режиме среднего разрешения (строки с 20 по 40). Далее производится ввод числа, характеризующего количество исходных значений (строки 60 и 70), и самих значений (строки с 80 по 120). В процессе ввода исходных значений в программе накапливается их общая сумма (строка 110).

После завершения ввода программа очищает экран дисплея и выводит заголовок (строки с 140 по 160). Затем выбирается начальная точка для первого сектора так, что ей соответствует 0 радиан (строка 170), и устанавливается значение радиуса окружности, равное 90 (строка 180).

В строках с 200 по 280 производится построение секторов круговой диаграммы. Для каждого сектора в программе вычисляются его концевая и средняя точки (строки 210 и 220). Затем на экран вблизи средней точки выводится соответствующее этому сектору исходное значение (строки 230 и 240), после чего строится сам сектор (строка 250). В используемом для этого операторе CIRCLE начальная и конечная точки задаются отрицательными значениями, вследствие чего на экране кроме дуги проводятся еще и оба радиуса. Для того чтобы значение, определяющее начальную точку, всегда было отрицательным (оно может оказаться равным нулю для самого первого сектора), из него вычитается .001. При выполнении строки 260 текущий сектор закрашивается, начиная с точки, расположенной приблизительно в центре сектора. В программной строке 270 устанавливается начальная точка для следующего сектора. Каждый последующий сектор начинается там, где кончается предыдущий.

```
10 DIM PART(25)
20 KEY OFF
30 SCREEN 1
40 CLS
50 '-- Ввод всех значений ----------------------------------
60 PRINT "Сколько отдельных частей";
70 INPUT N
80 FOR J=1 TO N
90 PRINT "Значение отдельной части";J;
100 INPUT PART(J)
110 TOTAL=TOTAL+PART(J) ' Подсчет общего количества
120 NEXT J
130 '-- Подготовка к построению круговой диаграммы ----------
140 CLS
150 LOCATE 1,15
160 PRINT TOTAL;"Итого";
170 STARTPT=0 ' Начальная точка для первого сектора
180 RAD=90 ' Радиус окружности
190 '--Построение на экране всех секторов--------------------
200 FOR J=1 TO N
210 ENDPT=6.283185*PART(J)/TOTAL+STARTPT'Вычисление координат конечн. точки
220 MIDPT=(STARTPT+ENDPT)/2 'Вычисление координат средней точки дуги сектора
230 LOCATE (100-SIN(MIDPT)*(RAD-8))/8,(160+COS(MIDPT)*(RAD+16))/8
240 PRINT PART(J); 'Вывод значения, соответст. данному сектору
250 CIRCLE (160,100),RAD,3,-STARTPT-.001,-ENDPT ' Построение сектора
260 PAINT (160+COS(MIDPT)*RAD*.75,100-SIN(MIDPT)*RAD*.75),
   J MOD 4,3 'Закрашивание сектора
270 STARTPT=ENDPT 'Нач. точка след. сектора совпадает с конечн. точкой текущего
280 NEXT J
290 LOCATE 22
300 END
```

Рис. 13.10. Программа построения круговой диаграммы.

После построения круговой диаграммы программа перемещает курсор в нижнюю часть экрана (строка 290), подготавливая его к выдаче сообщения «Ok», которое будет выводиться после завершения выполнения программы.

Язык графического вывода для расширенного Бэйсика

В расширенный Бэйсик входит оператор DRAW, который с помощью специального языка графического вывода позволяет изображать на экране рисунки, составленные из различных комбинаций точек и прямых линий. Начиная с последней высвеченной на экране точки, с помощью оператора DRAW можно провести линию любой длины и в любом из восьми заданных направлений. После этого можно провести еще одну линию, снова любой длины и в любом из восьми направлений. Подобный процесс можно продолжать

почти неограниченно. Каждая проводимая линия может быть окрашена в любой из четырех цветов активной палитры или делаться невидимой. Кроме того, можно проводить отрезок, соединяющий текущую точку с любой заданной абсолютными или относительными координатами точкой экрана. Можно также увеличивать или уменьшать весь выводимый рисунок, а также поворачивать его на 90, 180 или 270°.

Подкоманды графического вывода

Язык графического вывода, позволяющий проводить все указанные выше графические построения, состоит из 15 однобуквенных подкоманд. Составляя строки из этих подкоманд, можно определять различные строковые значения и тем самым задавать нужный рисунок. Для большинства подкоманд кроме идентифицирующих их букв необходимо также указывать некоторое число, означающее, например, какой длины проводить линию, какой использовать номер цвета, насколько увеличить весь рисунок и т. п.

Пример. Подкоманда R, проводящая линию вправо от данной точки.

```
10 SCREEN 1
20 DRAW "R159"
```

При выполнении этой программы на экране со средней разрешающей способностью будет проведена линия из центра экрана к правому его краю.

Здесь в операторе DRAW подкоманда R159 означает: «Начиная с текущей позиции провести линию на 159 столбцов вправо». Поскольку до выполнения каких-либо графических построений (в частности, сразу после выполнения оператора SCREEN) для интерпретатора Бэйсика ПВМ роль последней выведенной точки играет центр экрана, в данном примере на экране будет проведен отрезок, соединяющий точки (160,100) и (319,100).

Девять однобуквенных подкоманд обеспечивают вычерчивание линий на экране, одновременно осуществляя управление движением курсора, а остальные шесть реализуют специальные операции управления. В табл. 13.4 перечислены все 15 подкоманд графического вывода, а на рис. 13.11 показано, в каких направлениях проводятся линии при выполнении первых девяти подкоманд.

Комбинирование подкоманд графического вывода. Выполняя всего один оператор DRAW, можно получить на экране достаточно сложный рисунок, если командная строка в этом операторе состоит из нескольких подкоманд графического вывода, записанных в правильном порядке.

Таблица 13.4. **Подкоманды оператора DRAW**

Подкоманда	Действие	Значение параметров *n* [1]), *h*, *v*, *k* [1]), *s*$
	Подкоманды перемещения по экрану и вычерчивания линий [2])	
U*n*	Вверх	*n* — количество строк [3])
D*n*	Вниз	То же
R*n*	Вправо	*n* — количество столбцов [3])
L*n*	Влево	То же
E*n*	По диагонали вверх и вправо	*n* — расстояние по диагонали [4])
F*n*	По диагонали вниз и вправо	То же
G*n*	По диагонали вниз и влево	» »
H*n*	По диагонали вверх и влево	» »
M*h*,*v*	В заданную точку	*h*,*v* — абсолютные координаты
M +*h*, +*v*	То же	+*h*, +*v* — относительные координаты
	Управляющие подкоманды	
B	При последующем перемещении не проводить линию	Не задаются
N	После последующего перемещения возвратиться в текущую точку	То же
A*k*	Поворот против часовой стрелки всех последующих перемещений и проводимых линий [5])	*k* — угол поворота: 0 = 0° 2 = 180° 1 = 90° 3 = 270°
S*k*	Уменьшить или увеличить длины линий, которые будут проводиться в дальнейшем [5])	*k* — масштабный коэффициент: 1 = 1/4x 4 = 1x 8 = 2x 36 = 9x и т. д.
C*k*	Выбор цвета	*k* — номер цвета: 0 или 1 в режиме высокого разрешения; 0, 1, 2, 3 — в режиме среднего разрешения (цвета палитры)
X*s*$	Выполнить подкоманды другой строки	*s*$ — строковая переменная, содержащая дополнительные подкоманды

[1]) Для всех подкоманд, кроме M и X, значения параметров могут задаваться как целыми константами, так и числовыми переменными с целыми значениями. Для использования в подкоманде переменной необходимо поставить перед ее именем знак =, а после имени точку с запятой, например R=COL (R — подкоманда, а COL — имя переменной).

[2]) Если в подкоманде относительного перемещения объекта значение параметра выводит его за пределы экрана, то выполнение такой подкоманды завершается при достижении края экрана.

[3]) Расстояние в 5 ед. по вертикали равно расстоянию в 6 ед. по горизонтали. Например, для того чтобы изобразить на экране квадрат с вертикальной стороной в пять строк, его горизонтальную сторону необходимо проводить длиной в шесть столбцов. Заметим, однако, что при выполнении подкоманды диагонального перемещения диагонально противоположные углы такого квадрата соединяться не будут.

[4]) Единица измерения расстояния по диагонали по величине совпадает с 1,4142135 ед. расстояния по вертикали (количество строк). Например, диагональ, соединяющая два противоположных угла прямоугольника со сторонами 100 строк в высоту на 100 столбцов в ширину, имеет длину, равную 100 ед. по диагонали.

[5]) Действие подкоманд поворота и изменения масштаба не распространяется на подкоманду M, в которой используются абсолютные координаты.

Пример. Построение квадрата в центре экрана в режиме среднего разрешения.

```
40 DRAW "BM130,125U50R60D50L60" 'квадрат
```

В приведенном операторе DRAW подкоманде M130,125 предшествует подкоманда B, вследствие чего M130,125 становится командой только перемещения текущей точки, а никаких линий при

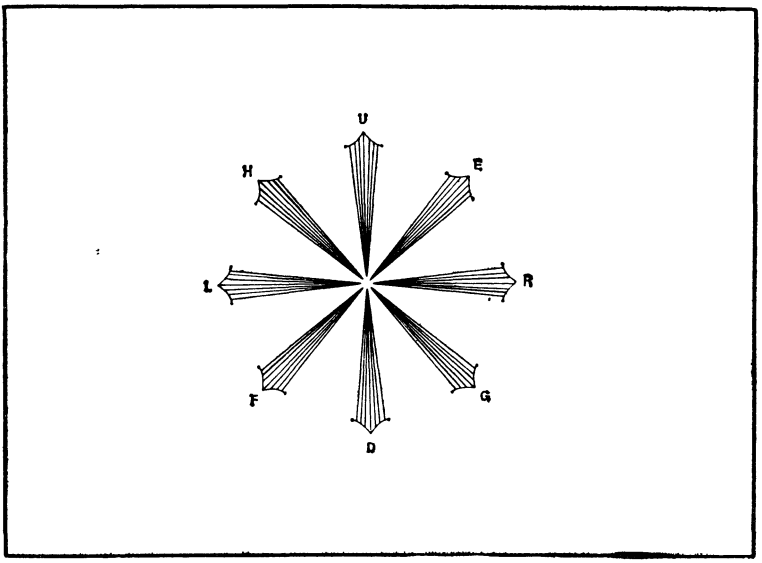


Рис. 13.11. Направление движения при выполнении различных подкоманд перемещения и проведения линий (подкоманды оператора DRAW).

ее выполнении не проводится. В соответствии с этим при выполнении следующей подкоманды U50 вертикальная линия на 50 строк вверх проводится из точки с координатами (130,125). После этого по подкоманде R60 проводится горизонтальная линия на 60 столбцов вправо, затем по подкоманде D50 — вертикальная линия на 50 строк вниз и, наконец, по L60 — горизонтальная линия на 60 столбцов влево, возвращающая нас в исходную точку с координатами (130,125). Несоответствие между количеством строк и количеством столбцов при построении квадрата связано с тем, что на экране, работающем в режиме средней разрешающей способности, шесть строк в высоту занимают столько же, сколько пять столбцов в ширину.

Задание числовых значений в подкомандах. Числовые значения подкоманд могут задаваться в виде числовых целых констант, стоящих непосредственно за символами соответствующих подко-

манд, как это было в двух предыдущих примерах, либо в виде числовых переменных. Во втором случае допустимыми являются элементы массивов, но не разрешается использовать функции и выражения. Исключением из общих правил является подкоманда M, в которой числовые значения можно задавать только с помощью целых констант, а переменные не допускаются. Для того чтобы в строке подкоманд вместо числовой константы использовать числовую переменную, необходимо перед именем этой переменной поставить символ =, а после имени — точку с запятой.

Пример

```
100 DRAW "u=VERT;"
```

В приведенном примере буква u обозначает подкоманду, а VERT — имя переменной. Заметим, что в строке подкоманд можно одновременно использовать и прописные, и строчные буквы.

Если для какой-либо подкоманды перемещения текущей точки и вычерчивания линии (кроме подкоманды M) задано числовое значение, выводящее за пределы экрана, то интерпретатор Бэйсика ПВМ завершит выполнение этой подкоманды при достижении края экрана. При этом, если значение не было отрицательным и не превышало 32767, сообщение об ошибке выдаваться не будет; для специальных управляющих подкоманд приняты иные ограничения. Так, целое значение, задаваемое для подкоманд A (угол поворота) и C (номер цвета), обязательно должно заключаться между 0 и 3, а значение, используемое в подкоманде S (масштабный коэффициент),— между 0 и 255; в противном случае появится сообщение об ошибке.

Использование специальных управляющих подкоманд. При выполнении программы, приведенной на рис. 13.12, на экране строится геометрическая фигура, изображенная на рис. 13.13. В процессе работы эта программа многократно выводит на экран один и тот же простой шестиугольник, изменяя с помощью подкоманды A угол его поворота, с помощью подкоманды C его цвет, а с помощью S размер. В программе иллюстрируется также использование подкоманды X, позволяющей в процессе выполнения некоторой строки подкоманд выполнять другие подкомандные строки.

В начальной части программы для экрана дисплея устанавливается режим работы со средней разрешающей способностью (строки с 10 по 30). Затем определяется строка подкоманд, с помощью которой на экране строится отдельный шестиугольник (строка 40). После этого следуют два вложенных друг в друга цикла FOR/NEXT, в которых изменяются масштаб, цвет и угол поворота выводимого на экран шестиугольника.

Во внешнем цикле (строки с 60 по 160) меняются масштаб и цвет очередного выводимого шестиугольника. Поскольку масштабный коэффициент, равный 4, соответствует воспроизведению рисунка

```
10 CLS:KEY OFF
20 SCREEN 1 'Режим среднего разрешения
30 COLOR 16,0 ' Черный фон, яркий передний план
40 HX$="u25e15r30d25g15l30" '1 шестиугольник
50 '--Вывод на экран при 10различных масштабных коэффициентах -
60 FOR SCALE=1 TO 10
70 S$="s"+STR$(SCALE) ' Масштабный коэффициент
80 '--Вычисление номера для заданного масштаба----------
90 C$="c"+STR$(SCALE MOD 3+1) ' Номер цвета
100 '-- Вывод шестиуг. при 4 различных углах поворота _____
110 FOR ROTATE=0 TO 3
120 R$="a"+STR$(ROTATE) ' Угол поворота
130 SHAPE$="xS$;xC$;xR$;xHX$;" ' Формир.строки подкоманд
140 DRAW SHAPE$ ' Вывод на экран отдельного шестиугольника
150 NEXT ROTATE
160 NEXT SCALE
170 A$=INPUT$(1)'Ожидание нажатия клавиши для оконч. работы
180 END
```

Рис. 13.12. Программа, использующая оператор DRAW.

без изменения его размеров, в программе используются значения коэффициента от 1 до 3 для изображения уменьшенных шестиугольников и от 5 до 10 — для увеличенных (строка 60). Для вычисления

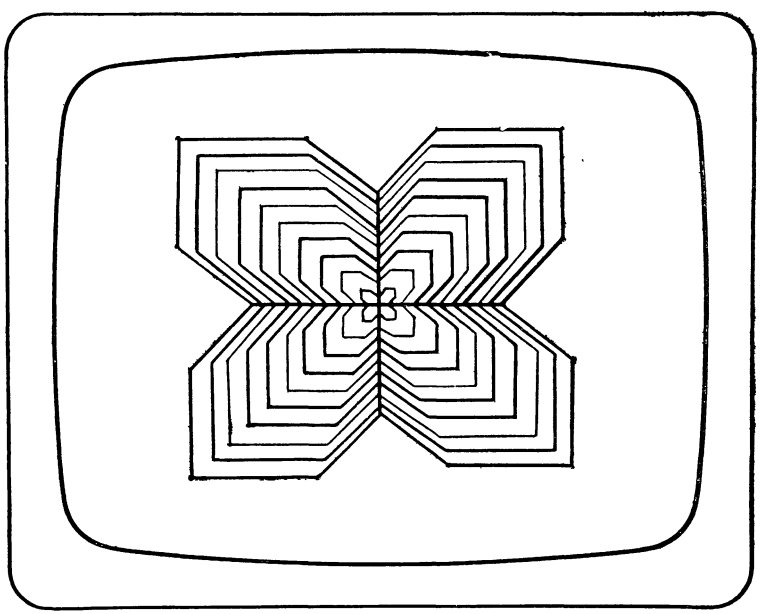


Рис. 13.13. Геометрическая фигура, составленная из шестиугольников (результат работы программы рис. 13.12).

очередного номера цвета к масштабному коэффициенту применяется операция MOD (вычитание по модулю), в результате чего при любом масштабном коэффициенте всегда получается одно из трех значений 0, 1 или 2 (строка 90). Для определения окончательного номера цвета к полученному значению прибавляются единица с тем, чтобы никогда не использовался номер, равный 0, который сответствует цвету фона, т. е. приводит к построению невидимых линий.

Во внутреннем цикле (строки 110—150) изменяется угол поворота. Для каждых зафиксированных во внешнем цикле масштабного коэффициента и номера цвета во внутреннем цикле на экран выводится шестиугольник при различных допустимых углах поворота. Задаваемые в программе коэффициенты поворота, равные 0, 1, 2 и 3, соответствуют повороту против часовой стрелки на 0, 90, 180 и 270°. При повороте на 90 или на 270° интерпретатор расширенного Бэйсика производит необходимую корректировку всех расстояний по вертикали и горизонтали так, чтобы сохранить исходные пропорции выводимого многоугольника.

Оператор DRAW, выполняющий собственно построение шестиугольника, помещается во внутреннем цикле (строка 140). Используемая в нем строка подкоманд SHAPE$ состоит из четырех различных командных строк: строка S$ устанавливает масштабный коэффициент для последующего вывода шестиугольника, строка C$ — цвет, R$ — угол поворота, а HX$ содержит подкоманды, выполняющие построение отдельного шестиугольника.

Синтез динамических изображений

«Оживление» изображения в кино (или мультипликация) в действительности основано на использовании набора обычных «неподвижных» картинок, каждая последующая из которых лишь очень незначительно отличается от предыдущей. Если последовательно просматривать эти картинки, очень быстро сменяя одну другой, то отдельные картинки будут сливаться друг с другом, создавая впечатление непрерывного плавного движения. При создании мультфильма с помощью кинокамеры снимают все отдельные картинки, нарисованные от руки или полученные каким-либо другим способом. После этого с помощью кинопроектора можно показывать фильм на экране.

Для синтеза динамического изображения на вычислительной машине можно также использовать кинокамеру и описанную выше технику: с помощью операторов PSET, LINE, CIRCLE, PAINT и DRAW можно рисовать на экране отдельные картинки и по очереди фотографировать их кинокамерой. Однако для вычислительной машины возможен более удобный подход к «оживлению» изображения, позволяющий обойтись и без кинокамеры, и без проектора, используя лишь возможности экрана дисплея.

Для того чтобы создать впечатление движущегося по экрану объекта, следует каждый раз сначала стирать его изображение с экрана, а затем снова выводить его на экран в новом месте. При этом местоположение объекта при каждом последующем выводе должно очень незначительно отличаться от его местоположения при предыдущем выводе, иначе вместо плавного перемещения будет лишь видно, как изображение объекта исчезает в одном месте, а затем появляется в другом. Все это нетрудно реализовать с помощью программы, организуя всего один простой цикл и используя рассмотренные ранее операторы графического вывода. Однако для того чтобы движение изображения не было «дергающимся», отдельные кадры должны очень быстро сменять друг друга. Используя только обычные операторы графического вывода, это требование выполнить трудно, если вообще возможно. В связи в этим в расширенный Бэйсик включены два дополнительных оператора графического вывода GET и PUT, позволяющие решить указанную проблему.

«Оживление» изображения можно использовать в развлекательных целях. Даже отдельный объект может оказаться достаточно занимательным, если перемещать его по экрану, меняя его размер и скорость движения. Кроме того, движущееся изображение имеет и практическое применение. Примерами являются перемещение указателей по диаграмме или графику, а также последовательное перемещение при показе внутренних разрезов того или иного устройства для объяснения механизмов его работы.

Операторы GET и PUT

Оператор GET позволяет запоминать цвета всех точек заданной прямоугольной области экрана дисплея и хранить их номера в виде числового массива; оператор PUT повторно воспроизводит все эти цвета на экране. Можно считать, что оператор GET играет роль кинокамеры, числовой массив является слайдом, а оператор PUT как бы выполняет функции проектора слайдов. С помощью оператора PUT изображение воспроизводится достаточно быстро для того, чтобы можно было синтезировать движение объектов по экрану.

Определение области экрана для оператора GET. Прямоугольная область экрана, раскраска которой должна запоминаться при выполнении оператора GET, может быть любого размера и располагаться в любом месте экрана. В качестве примера рассмотрим следующую программу, которая строит в середине экрана некоторую картину взрыва (рис. 13.14):

```
10 '— — Темно-синий фон и — — — —
20 'светлая палитра переднего плана с цветами зеленый/красный/
   золотистый
30 CLS:KEY OFF:SCREEN 1:COLOR 17,0
```

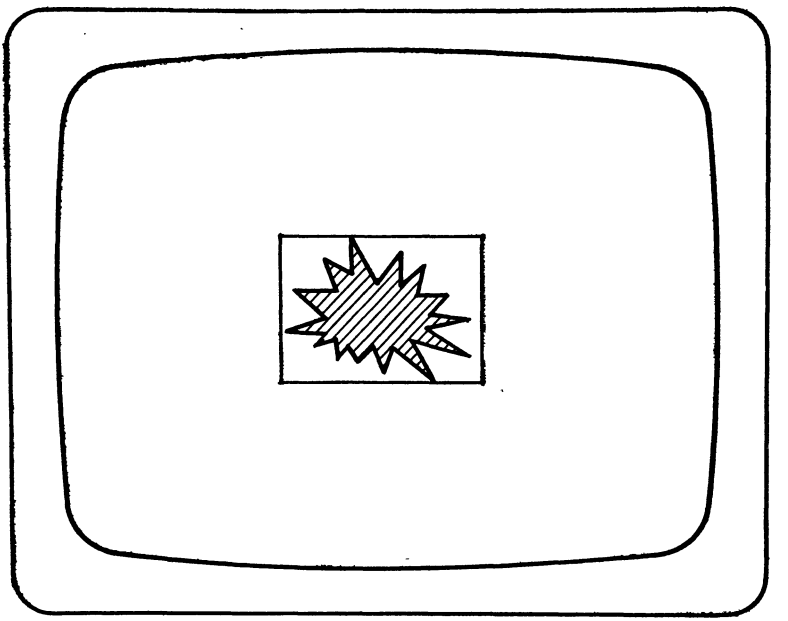


Рис. 13.14. Изображение картины взрыва.

```
40 '— — Построение картины взрыва — —
50 DRAW "c2bm190,82m178,91m178,73m163,91
   m148,65m148,85m133,79m139,94m112,94
   m133,109m106,118m133,118m127,124m139,121
   m139,130m145,124m151,133m160,124m166,139
   m172,124m196,145m184,121m223,133m193,115
   m217,109m196,106m205,97m187,97m190,82"
60 '— — Закрашивание золотистым цветом
70 PAINT (160,100),3,2
80 '— — Выделение занятой области экрана
100 LINE (105,65)—(225,145),1,B
```

Форма, размер, местоположение прямоугольной области, информация о которой должна запоминаться в операторе GET, задаются координатами двух ее противоположных углов, т. е. точно так же, как в операторе LINE с опцией B задается подлежащий построению прямоугольник. В приведенном выше примере с помощью оператора LINE область экрана, реально занятая картиной взрыва, обводится прямоугольником. Эта прямоугольная область с координатами противоположных углов (105,65) и (225,145) и составляет все, что нужно обработать оператору GET для запоминания данной картины взрыва.

Вычисление размера массива для оператора GET. Для выполнения оператора GET необходимо задать числовой массив достаточно большой для того, чтобы вместить данные о расцветке всех точек рассматриваемой области экрана. На рис. 13.15 приводится простая программа, подсчитывающая минимальное количество элементов для такого массива числовых значений,— целых, с обычной или с двойной точностью. В начале работы программа выдает запрос на ввод координат, которые будут задаваться в операторе GET, а также просит указать, какой режим графического вывода будет использоваться. Если, например, в качестве исходных данных задать координаты прямоугольника, который строился в предыдущем примере

```
10 CLS
20 INPUT "Режим среднего разрешения (Y/N)"; R$
30 IF R$="Y" OR R$="y" THEN BR=2: GOTO 60
40 INPUT "Режим высокого разрешения (Y/N)"; R$
50 IF R$="Y" OR R$="y" THEN BR=1 ELSE GOTO 10
60 INPUT "Координаты первой точки (столбец, строка)",H1,V1
70 INPUT "Координаты второй точки (столбец, строка)",H2,V2
80 BYTES=4+INT(((ABS(H1-H2)+1)*BR+7)/8)*(ABS(V1-V2)+1)
90 PRINT "--Минимальные размеры массива
100 PRINT "INTEGER%(";CINT(BYTES/2)-1;")"
110 PRINT " SINGLE!(";CINT(BYTES/4)-1;")"
120 PRINT " DOUBLE#(";CINT(BYTES/8)-1;")"
130 PRINT "(";BYTES;" байт требуется)"
```

Рис. 13.15. Программа вычисления размера массива для оператора GET.

для выделения области экрана, занятой картиной взрыва, то данная программа определит, что задаваемый для оператора GET массив значений с обычной точностью должен состоять не менее чем из 628 элементов.

Пример. Оператор определения массива для запоминания картины взрыва из предыдущего примера, а также использующий этот массив оператор GET.

```
90 DIM XPLODE(628)
100  GET  (105,65)—(225,145),XPLODE
```

Заметим, что в операторе GET указывается только имя массива без скобок и индексов.

Воспроизведение на экране хранимого в памяти изображения. Оператор PUT позволяет воспроизводить в любом месте на экране все, что ранее содержалось в выделенной прямоугольной области и запомнилось при выполнении оператора GET (в форме сведений о раскраске области). Для этого достаточно задать в операторе PUT координаты точки, в которой должен располагаться верхний левый угол воспроизводимого прямоугольника, и имя числового массива, содержащего данные о раскраске этого прямоугольника.

Пример. Оператор PUT, реализующий картину взрыва, изображенную на рис. 13.14; здесь эта картина воспроизводится дважды, каждый раз на новом месте.

```
110 '— —Двукратное воспроизведение изображения— — — — —
120 PUT (15,25),XPLODE
130 PUT (199,0),XPLODE
```

Чтобы стереть с экрана изображение, полученное с помощью оператора PUT, подобного вышеприведенному, достаточно еще раз повторить тот же самый оператор PUT. Для предыдущего примера это выглядит следующим образом:

```
150 '— — Стирание всех трех изображений — — — — — — —
160 PUT (105,65),XPLODE '1-е (исходное) изображение
170 PUT (15,25),XPLODE 'Стирание обеих
180 PUT (199,0),XPLODE 'копий
```

Движущиеся объекты

Для синтезирования на экране движения объекта, данные об изображении которого хранятся в числовом массиве, необходимо выполнить следующие действия:

- воспроизвести изображение объекта на экране с помощью оператора PUT (подобного рассмотренному выше);
- вычислить координаты нового местоположения объекта на экране;
- стереть с помощью оператора PUT текущее изображение объекта на экране;
- с помощью оператора PUT повторно воспроизвести изображение объекта, но с новыми координатами (в новом месте экрана);
- повторить все шаги, начиная со второго.

Пример. «Оживление» картины взрыва путем многократного вывода ее в различных местах экрана.

```
190 '— — Динамическая картина взрыва — — — — — — — —
210 NEWCOL=0:NEWROW=119 'Начальное местоположение
220 PUT (NEWCOL, NEWROW),XPLODE 'Воспроизведение
    изображения
230 FOR DUP=1 TO 10
240 OLDCOL=NEWCOL:OLDROW=NEWROW
250 NEWCOL=RND*199:NEWROW=RND*119
260 FOR T%=1 TO 2100/10:NEXT 'Пауза
270 PUT (OLDCOL,OLDROW),XPLODE 'Стирание
    изображения
280 PUT (NEWCOL,NEWROW),XPLODE 'Воспроизведение
    изображения
300 NEXT DUP
```

В 260-й строке приведенного программного сегмента выполняется пустой цикл FOR/NEXT, за счет чего каждый раз перед стиранием и повторным воспроизведением изображения выдерживается пауза длительностью около 1/10 с. Число повторений этого цикла выбрано на основе экспериментальных данных, которые показывают, что интерпретатор Бэйсика ПВМ выполняет пустой цикл FOR/NEXT со скоростью приблизительно 2100 раз в секунду.

Координаты каждого нового местоположения картины взрыва в данном программном сегменте определяются с помощью функции RND (строка 250). Результатом выполнения этой функции является случайная десятичная дробь, заключенная между 0 и 1, так что умножение значения функции на 199 или 119 дает случайное число, заключенное между 0 и 199 или между 0 и 119 соответственно. Однако при каждом прогоне программы функция RND будет генерировать одну и ту же последовательность чисел. Чтобы использовать действительно случайную последовательность при всяком прогоне программы, можно добавить в исходный программный сегмент следующий оператор:

```
200 RANDOMIZE VAL(RIGHT$(TIME$,2))
```

Оператор RANDOMIZE, подобный приведенному выше, генерирует набор случайных чисел, из которого в дальнейшем выбирают значения функции RND. Задаваемое в этом операторе значение определяет, какой именно набор случайных чисел будет генерироваться: каждое целое число из интервала от —32768 до 32767 определяет свой, отличный от всех других набор. Для произвольного выбора целого значения в данном примере оно задается с помощью функции TIME$. Эта функция, имеющаяся только в дисковом и в расширенном Бэйсике, выдает текущее время суток в соответствии с показаниями встроенных в ПВМ «часов». Время выдается в виде восьмисимвольного строкового значения вида «08:30:11». В приведенном выше примере используются только два последних символа такой строки, т. е. значение секунд, которое всегда представляет собой целое число, заключенное между 0 и 59.

Возможные параметры оператора PUT

При выполнении оператора PUT простейшего вида на экране в общем случае появляются не те цвета, номера которых хранятся в заданном для этого оператора числовом массиве, а цвета, представляющие собой результат взаимодействия заданных с текущей раскраской рассматриваемой области экрана. Это обстоятельство уже использовалось в предыдущих примерах, когда для стирания некоторого изображения оно еще раз в том же самом виде воспроизводилось на экране с помощью оператора PUT, при этом оно накладывалось само на себя, и не затрагивало фона. В действительности существуют пять различных способов взаимодействия изоб-

ражения, воспроизводимого оператором PUT, с текущим изображением на экране дисплея. Конкретный способ выбирается путем задания в конце оператора PUT одного из пяти ключевых слов: PSET, PRESET, AND, OR или XOR, например так:

```
280 PUT (NEWCOL,NEWROW),XPLODE,XOR
```

Оператор PUT, подобный приведенному выше, т. е. оканчивающийся необязательным параметром XOR, выполняется точно так же, как простой PUT без каких-либо параметров в конце. Параметры AND, XOR и OR определяют три различных набора правил, в соответствии с которыми происходит слияние заданных в операторе PUT цветов с текущей расцветкой экрана. Все эти правила представлены в табл. 13.5. Например, если в операторе PUT задан

Таблица 13.5. **Параметры слияния цветов для оператора PUT [1])**

Заданный номер цвета	Цвет экрана при суффиксе AND				Цвет экрана при суффиксе OR				Цвет экрана при суффиксе XOR			
	0	1	2	3	0	1	2	3	0	1	2	3
0	0	0	0	0	0	1	2	3	0	1	2	3
1	0	1	0	1	1	1	3	3	1	0	3	2
2	0	0	2	2	2	3	2	3	2	3	0	1
3	0	1	2	3	3	3	3	3	3	2	1	0

[1]) Существуют еще два других оператора слияния цветов: PSET и PRESET. Первый из них всегда воспроизводит в точности заданные цвета (цвет экрана игнорируется); второй воспроизводит цвет, обратный по отношению к заданному (взаимно обратными считаются цвета с номерами 0 и 3, а также с номерами 1 и 2).

параметр AND, то при воспроизведении этим оператором голубого цвета (с номером 1) в точке экрана, окрашенной в данный момент в белый цвет (номер 3), получается голубая точка. Если выполняется PUT с параметром OR, то при слиянии голубого и белого образуется белый цвет, а если в PUT задан параметр XOR, то результатом слияния этих же цветов является пурпурный цвет.

При выполнении оператора PUT с параметром PSET не происходит никакого слияния заданного цвета с текущим цветом экрана: независимо от раскраски текущего изображения на экране после выполнения такого PUT каждая точка окрашивается в точности в тот цвет, который фиксировался и запоминался оператором GET. В случае параметра PRESET, точно так же как и при PSET, игнорируется текущая окраска точек экрана, но при этом вместо заданного цвета воспроизводится «противоположный» ему: принято, что противоположными друг другу являются цвета с номерами 0 и 3, а также цвета с номерами 1 и 2.

Пример синтеза динамического изображения

В программе, приведенной на рис. 13.16, синтезируется движение по экрану дисплея десяти объектов: девяти точек и одного треугольника. Происходящий на экране процесс движения иллюстрируется на рис. 13.17 отдельными картинками, соответствующими изображению в различные моменты времени. Сначала по экрану

```
 9 '-- Опред. массивов графич. вывода для операторов GET и PUT ---
 10 DIM DOT(37),WEDG1(82),WEDG2(82),WEDG3(82),WEDG4(82),WEDG5(82)
 20 DOT.HITE=21
 30 DIM NOP(8,25)'Для хранения очередных горизонтальных позиций точек
 90 '-- Построение объектов ----------------------------------
 100 CLS:KEY OFF:SCREEN 1:COLOR 0,1
 110 CIRCLE (160,100),10,1,0,6.283001
 120 PAINT (160,100),1,1
 130 GET (148,90)-(172,110),DOT
 190 '-- Закрытый треугольник ------------
 200 CLS:DRAW "C2BM142,100M+30,-20M+0,+40M-30,-20"
 210 PAINT (160,100),2,2
 220 GET (142,80)-(172,120),WEDG1
 290 '-- Частично раскрытый треугольник ---
 300 ANGL$="M-30,-10M+30,-10" Подкоманды для оператора
 310 CLS:DRAW "C2BM172,120;XANGL$;XANGL$;M+0,+40;"
 320 PAINT (171,101),2,2:PAINT (171,99),2,2
 330 GET (142,80)-(172,120),WEDG2
 390 '-- Полностью раскрытый треугольник --
 400 CLS:DRAW "C2BM142,80M+30,+0M+0,+40M-30,+0"
 410 GET (142,80)-(172,120),WEDG3
 490 '-- Узкий треугольник ---------------
 500 CLS:DRAW "C2BM142,100M+30,-10M+0,+20M-30,-10"
 510 PAINT (160,100),2,2
 520 GET (142,80)-(172,120),WEDG4
 590 '--Треугольник вместе с точкой--------
 600 CLS:PUT (142,80),WEDG1:PUT (148,90),DOT
 610 GET (142,80)-(172,120),WEDG5
 990 '--Получение горизонтальных координат точек при всех возм. состояниях ----
 1000 CLS:FOR FRAME=0 TO 25:FOR K%=0 TO 8
 1010 READ NOP(K%,FRAME)
 1020 NEXT K%,FRAME
 1190 '-- Движение точек по экрану----------------------------
 1200 GOSUB 4900 ' Появление точек
 1290 '--Пятикратное повт. волнооб. движения выстроенных в линию точек
 1300 FOR DUP=1 TO 5:FOR FRAME=1 TO 2
 1310 GOSUB 4000 ' Перемещение точек
 1320 NEXT FRAME,DUP
 1390 '-- Появление треугольника-----------------------------
 1400 OLDCOL=285:NEWCOL=OLDCOL:ROW=3.5*DOT.HITE
```

Рис. 13.16. Пример программы синтеза динамического изображения.

движутся выстроенные в линию точки. Затем линия начинает совершать волнообразные движения, напоминая движения ленты при небольшом ветре. Вскоре после этого в правом конце экрана появляется треугольник, пульсирующий в результате раскрытия и быстрого сжатия угла при вершине. Далее он стремительно продвигается по направлению к линии точек и врезается в нее, но точки при этом оказывают сопротивление. В конце концов треугольник прорывает линию, выбивая из нее одну точку, которая быстро отле-

```
1410 PUT (OLDCOL,ROW),WEDG1
1420 FOR DUP=1 TO 2
1430 GOSUB 10000:GOSUB 4600 ' Колебание треугольника после средней задержки
1440 NEXT DUP
1450 GOSUB 10000 ' Средняя задержка
1490  -- Приближение треугольника к точкам ----------
1500 FOR NEWCOL=OLDCOL-15 TO NDP(4,2)+24 STEP -15
1510 GOSUB 4500 ' Перемещение треугольника
1520 NEXT
1590 '-- Сопротивление точек натиску треугольника ----------
1600 NEWCOL=OLDCOL 'Начинается с текущей позиции треугольника
1610 FOR DUP=1 TO 2
1620 FOR FRAME=3 TO 10 '3 шага вперед  , 4 назад 1 вперед
1630 IF FRAME<6 OR FRAME=10 THEN NEWCOL=NEWCOL-5 ELSE NEWCOL=NEWCOL+5
1640 GOSUB 4000:GOSUB 4500' Перемещение точек, а затем-треугольника
1650 NEXT FRAME, DUP
1790 '-- Треугольник прорывается сквозь линию точек ---------
1800 FOR FRAME=11 TO 18
1810 NEWCOL=NEWCOL-5
1820 GOSUB 4000:GOSUB 4500'Линия точек разрывается и треуг. продвиг.  вперед
1830 NEXT FRAME
1840 GOSUB 10000 ' Средняя задержка
1850 GOSUB 4600 ' Колебания треугольника
1860 PUT(OLDCOL,ROW),WEDG1' Сокращение
1870 PUT(OLDCOL,ROW),WEDG4'треугольника
1890 '--Треугольник продвиг. вперед и проходит через образов. разрыв. в линии точек
1900 FOR NEWCOL=OLDCOL TO 65 STEP -5
1910 PUT (OLDCOL,ROW),WEDG4
1920 PUT (NEWCOL,ROW),WEDG4
1930 OLDCOL=NEWCOL
1940 NEXT NEWCOL
1950 PUT (OLDCOL,ROW),WEDG4 ' Восстановление обычного
1960 PUT (OLDCOL,ROW),WEDG1 ' размера треугольника
1970 GOSUB 10000 ' Средняя задержка
1980 PUT (OLDCOL,ROW),WEDG1 ' Стирание изображения треугольника
2090 '-- Захват треуг. выбитой точки ----------
2100 GOSUB 4700 '  Частичное раскрытие треугольника
2110 PUT (OLDCOL,ROW),WEDG3 ' Полное раскрытие треугольника
2120 FOR NEWCOL=OLDCOL TO 35 STEP -5 ' Захват точки
2130 PUT (OLDCOL,ROW),WEDG3
```

Рис. 13.16. (Продолжение.)

тает к противоположному концу экрана. После этого треугольник снова сокращается, уменьшаясь до размеров, позволяющих ему пройти через образовавшийся в линии проход, и врезается в линию точек, прорываясь сквозь нее. Затем треугольник полностью раскрывается, захватывает выбитую точку и снова закрывается. Точки опять выстраиваются в линию. Треугольник вместе с захваченной им точкой удаляется с экрана. Выбитая точка заменяется на новую, а затем вся линия также удаляется с экрана.

```
2140 PUT (NEWCOL,ROW),WEDG3
2150 OLDCOL=NEWCOL
2160 NEXT NEWCOL
2170 PUT (OLDCOL,ROW),WEDG3 'Стирание раскрытого треугольника
2190 '--Закрытие треугольника ----------
2200 GOSUB 4700 'Частичное закрытие треугольника
2210 PUT (OLDCOL,ROW),WEDG1 'Полное закрытие треугольника
2290 '--Выпрямление линии точек----------
2300 FOR FRAME=19 TO 25
2310 GOSUB 4000 'Перемещение точек
2320 NEXT FRAME
2390 '-Удаление треугольника с экрана (полностью)----------
2400 FOR NEWCOL=OLDCOL TO 0 STEP -3
2410 PUT (OLDCOL,ROW),WEDG5
2420 PUT (NEWCOL,ROW),WEDG5
2430 OLDCOL=NEWCOL
2440 NEXT NEWCOL
2450 PUT (OLDCOL,ROW),WEDG5 'Стирание треугольника
2460 GOSUB 10000:GOSUB 10000 'Две средние задержки
2490 '-- Замена захвач.точки на новую и удаление с экрана всей линии точек
2500 PUT (140,4*DOT.HITE),DOT 'Замена захваченной точки
2510 GOSUB 10000:GOSUB 10000 'Две средние задержки
2520 GOSUB 4900 ' Удаление точек
3000 END
3990 '=Перемещение всех точек ==========
4000 FOR K%=0 TO 8
4010 IF NDP(K%,FRAME-1)<>NDP(K%,FRAME) THEN PUT (NDP(K%,FRAME-1),
     K%*DOT.HITE),DOT:PUT(NDP(K%,FRAME),K%*DOT.HITE),DOT
4020 NEXT K%:RETURN
4490 '==Перемещение треугольника==========
4500 PUT (OLDCOL,ROW),WEDG1
4510 PUT (NEWCOL,ROW),WEDG1
4520 OLDCOL=NEWCOL
4530 RETURN
4590 '==Колебания треугольника==========
4600 FOR K%=1 TO 5
4610 PUT (OLDCOL,ROW),WEDG1
4620 GOSUB 4700 'Частичное раскрытие треугольника
4630 PUT (OLDCOL,ROW),WEDG1 'Восстановл. обычного размера треугольника
4640 NEXT K%:RETURN
```

Рис. 13.16. (Продолжение.)

Для синтеза такого небольшого фрагмента используется шесть отдельных объектов (рис. 13.18): одна точка и пять разновидностей треугольника, включающих в себя обычный треугольник, полураскрытый и полностью раскрытый треугольник для синтеза быстро открывающегося и захлопывающегося треугольника, узкий треугольник для прохождения сквозь линию точек, а также треугольник вместе с захваченной им точкой. В самом начале каждый

```
4690 '=Частично раскрытый треугольник==========
4700 PUT (OLDCOL,ROW),WEDG2
4710 PUT (OLDCOL,ROW),WEDG2
4720 RETURN
4890 '=Появление точек на экране или удаление их с экрана=
4900 FOR K%=0 TO 8
4910 PUT (NDP(K%,0),K%*DOT.HITE),DOT
4920 GOSUB 10100 ' Короткая задержка
4930 NEXT K%:RETURN
9990 '=Временная задержка средней длительности =======
10000 FOR T%=1 TO 1250*1.5:NEXT:RETURN
10090 '=Временная задержка короткой длительности======
10100 FOR T%=1 TO 1250/8:NEXT:RETURN
10990 '-- Горизонтальные координаты точек-----------------
20000 DATA 140,140,140,140,140,140,140,140,140 :REM Начальные позиции
20010 DATA 130,130,130,130,130,130,130,130,130 :REM Позиции при волнообразном
20020 DATA 140,140,140,140,140,140,140,140,140 :REM движении
20030 DATA 140,140,140,140,135,140,140,140,140 :REM Позиции при оказании
20040 DATA 140,140,140,135,130,135,140,140,140 :REM сопротивления треугольнику
20050 DATA 140,140,135,130,125,130,135,140,140
20060 DATA 140,140,140,135,130,135,140,140,140
20070 DATA 140,140,140,140,135,140,140,140,140
20080 DATA 140,140,140,140,140,140,140,140,140
20090 DATA 140,140,140,140,145,140,140,140.140
20100 DATA 140,140,140,140,140,140,140,140,140
20110 DATA 140,140,140,140,135,140,140,140,140 :REM Позиции при прохождении
20120 DATA 140,140,140,135,130,135,140,140,140 :REM треугольника сквозь
20130 DATA 140,140,135,130,125,130,135,140,140 :REM линию точек
20140 DATA 140,135,130,125,110,125,130,135,140
20150 DATA 140,135,125,115,090,115,125,135,140
20160 DATA 140,135,125,105,065,105,125,135,140
20170 DATA 140,135,125,105,050,105,125,135,140
20180 DATA 140,135,125,105,041,105,125,135,140
20190 DATA 140,135,125,115,041,115,125,135,140 :REM Позиции при выпрямлении
20200 DATA 140,135,130,125,041,125,130,135,140 :REM оставшихся точек
20210 DATA 140,140,135,130,041,130,135,140,140
20220 DATA 140,140,140,135,041,135,140,140,140
20230 DATA 140,140,140,140,041,140,140,140,140
20240 DATA 140,140,140,145,041,145,140,140,140
20250 DATA 140,140,140,140,041,140,140,140,140
```

Рис. 13.16. (Продолжение.)

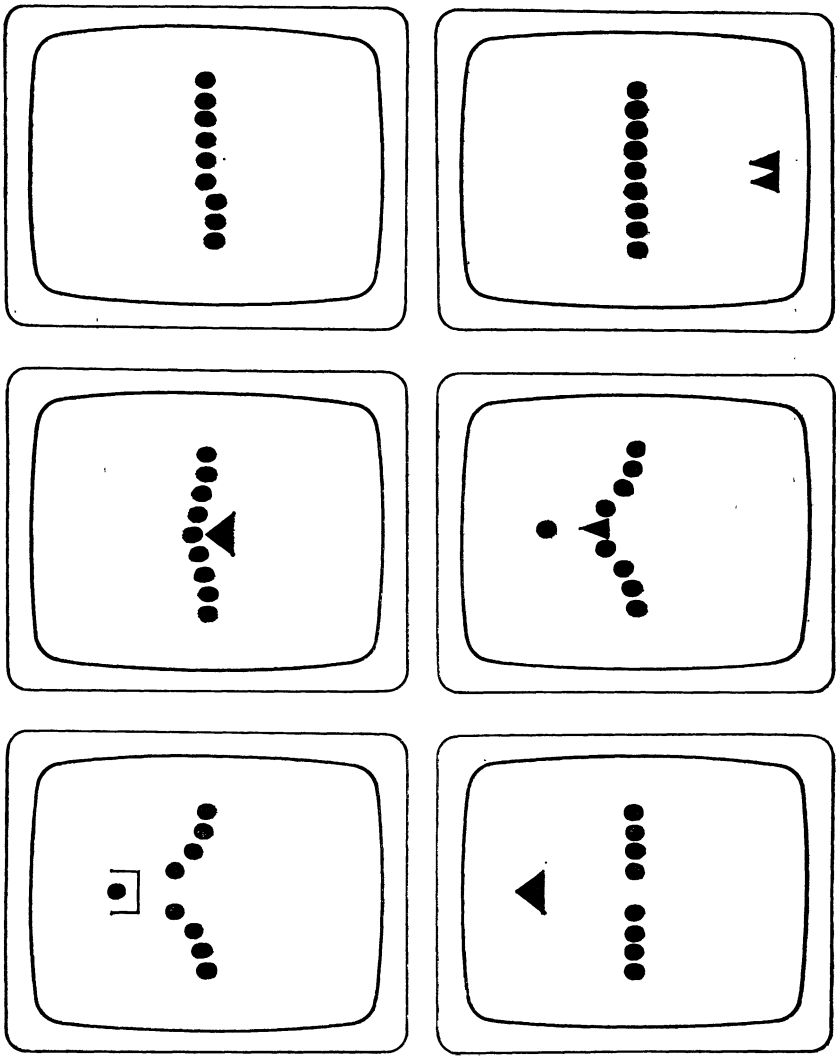


Рис. 13.17. Состояние экрана в различные моменты времени при выполнении программы синтеза динамического изображения, приведенной на рис. 13.16.

из указанных объектов на очень короткое время «вспыхивает» на экране; в программе при этом производится вывод соответствующего объекта на экран и запоминание его изображения с помощью оператора GET.

Работа программы поясняется имеющимися в ней комментария-

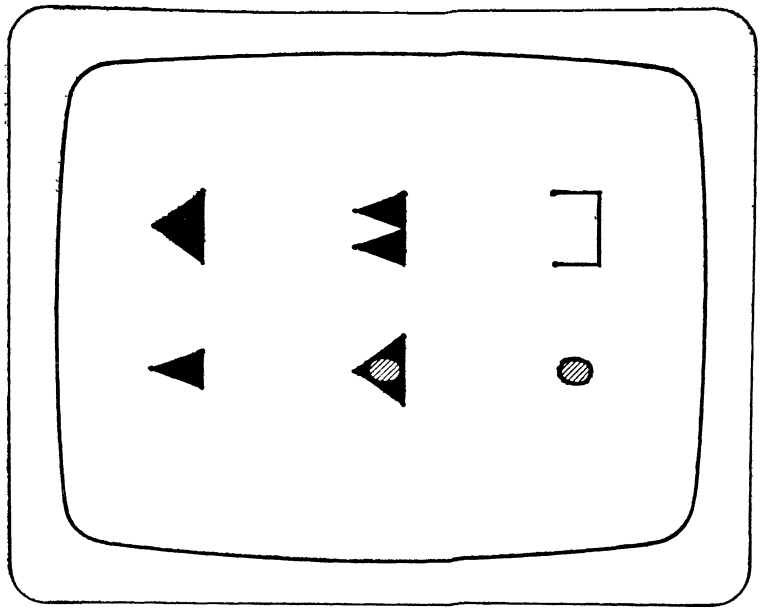


Рис. 13.18. Объекты, используемые в программе синтеза динамического изображения, приведенной на рис. 13.16 и рис. 13.17.

ми. Обратим внимание на то, как операторы DRAW строят различные виды треугольника, используя при этом относительные координаты (строки 200, 310, 400, 500), а в одном случае — подкоманду X для включения в текущую строку новых подкоманд (строки 300 и 310). Для задания горизонтальных позиций каждой точки в различные моменты ее движения в программе используется массив, в который заносятся значения из списка значений операторов DATA (строки с 20000 по 20250). Существуют и другие способы задания позиций изображаемого на экране объекта; например, для треугольника позиция вычисляется всякий раз, когда нужно изменить его положение.

Наблюдая за изображением на экране в процессе работы программы, нетрудно заметить, что объекты в разные моменты передвигаются с разной скоростью. Линия точек при волнообразных движениях двигается быстрее, чем когда она прогибается под нажимом треугольника, поскольку каждое отдельное расстояние, проходимое точками в процессе волнообразного движения, вдвое больше аналогичного расстояния при прогибании линии. Например, пятая точка линии при волнообразных движениях перемещается между 140-м и 130-м столбцами (строки 20010 и 20020), а прогибается она, начиная со 140-го и кончая 135-м столбцом (строки 20020 и 20030).

Скорость движения треугольника изменяется в основном по другой причине. Он перемещается быстро, когда является единственным движущимся объектом (например, строки с 1500 по 1520), но, если в то же самое время на экране двигаются и точки, он заметно снижает свою скорость (см., например, строки с 1610 по 1650). Если на экране одновременно должны двигаться 10 объектов, то для отдельного объекта между двумя последовательными изменениями его положения будет проходить больший интервал времени.

Глава 14

ВОСПРОИЗВЕДЕНИЕ ЗВУКА

В персональных ЭВМ можно генерировать звуки и музыку при помощи встроенных динамиков, используя один из двух операторов SOUND или PLAY. Оба оператора дают возможность управлять частотой и продолжительностью звука, но не его громкостью. Всегда генерируются только чистые тона; непосредственного способа их целевого искажения для создания различных акустических эффектов не существует.

Генерация звуков

При помощи оператора SOUND можно генерировать звук любой частоты в диапазоне 37 — 32 767 Гц длительностью от доли секунды до получаса.

Пример

```
SOUND 532.25, 18.2
```

В данном примере генерируется звук частотой 532.25 Гц, которому на нотном стане соответствует нота «до» первой октавы (или в буквенном обозначении С). На рис. 14.1 приведены высота звуков, образующих две октавы выше и две октавы ниже первой октавы.

Второе число в операторе SOUND определяет длительность звучания, которая измеряется *числом импульсов сигнала времени* и составляет в данном примере 18,2 имп./с. В табл. 14.1 сравнивается ряд значений частот импульсов времени с типичными темпами исполнения музыкальных произведений и эквивалентным каждому из них числом ударов метронома в 1 мин. В Бэйсике ПВМ не предполагается ожидание окончания работы оператора SOUND для перехода к выполнению очередного оператора.

Пример

```
10 CLS:WIDTH 40
20 SOUND 440, 27.3
30 FOR K=1 TO 40
40 LOCATE RND(1)*23+1,RND(1)*39+1
50 PRINT CHR$(14);
60 NEXT K
70 LOCATE 24,1
```

После очистки экрана приведенная выше программа генерирует звук «ля» малой октавы, имеющий длительность $1^1/_2$ с. Пока ПВМ «держит» эту ноту, выполняется оставшаяся часть программы, ко-

торая обеспечивает вывод изображения нот в произвольные места экрана. Фактически работа программы завершается раньше, чем отзвучит нота.

Бэйсик ПВМ не предусматривает перекрытия двух операторов SOUND. Если выполнение второго такого оператора должно на-

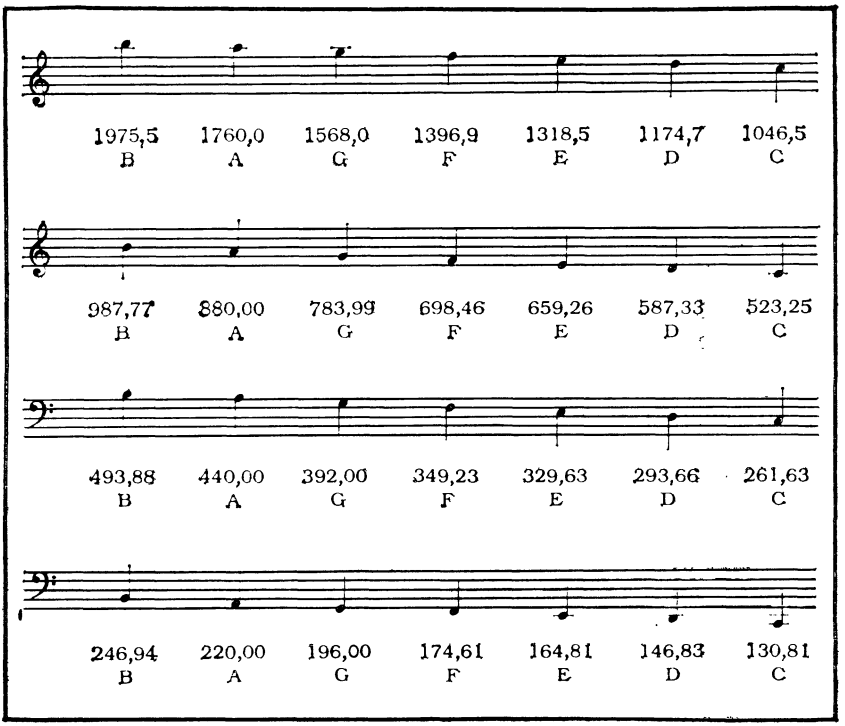


Рис. 14.1. Частоты музыкальных нот.

чаться до окончания звука, сгенерированного предыдущим оператором SOUND, ПВМ ожидает окончания этого звука. Например, если приводимую ниже строку добавить к предыдущему примеру, то звук, который в ней генерируется (нота «ре» первой октавы), не должен зазвучать до тех пор, пока не закончится предыдущий звук длительностью $1^1/_2$ с:

```
80 SOUND 587.33, 9.1
```

В любой момент звук можно убрать, выполнив оператор SOUND с нулевой длительностью.

Таблица 14.1. **Темп в музыке**

Число имп. сигнала врем. в 1с	*Темп*	*Число ударов метронома в 1 мин*
	Larghissimo	
27.30		40
	Largo	
18.20		60
	Larghetto	
	Grave	
	Lento	
16.55		66
	Adagio	
	Adagietto	
14.37		76
	Andante	
	Andantino	
10.11		108
	Moderato	
	Allegretto	
9.10		120
	Allegro	
	Vivace	
6.50		168
	Presto	
5.25		208
	Prestissimo	

Пример

```
sound 1760,32767
Ok
sound 100,0
Ok
```

Звуки, имеющие частоту выше 25 000 Гц, неслышимы; фактически большинство людей не воспринимают звуки частотой более 15 000 Гц. Следовательно, оператор SOUND, в котором указана частота более 15 000 Гц, будет генерировать паузы.

Звуковые эффекты

Оператор SOUND можно использовать также для создания всевозможных звуковых эффектов. К сожалению, не существует ни руководств, ни правил, которые можно было бы здесь применить, и все, что можно сделать, это провести эксперимент. Для начала можно использовать программу, несколько строк которой приведены на рис. 14.2.

Музыка

Несмотря на то что с помощью оператора SOUND можно исполнять музыкальные произведения, необходимый при этом перевод нот в форму многоразрядных чисел оказывается в лучшем случае

```
10 REM !-- Случайный шум ---------->
20 SOUND RND(1)*300+440,RND(1)*RND(1)
30 GOTO 20

10 '-- Глухие удары ------------------>
20 FOR K=60 TO 1 STEP -2
30 SOUND 246.94-K/2,K/20
40 SOUND 32767,K/15
50 NEXT K

10 '-- Затихание мелодии ------------->
20 FOR K=2000 TO 550 STEP -10
30 SOUND K,K/4000
40 NEXT K

10 '-- Сирена ------------------------>
20 FOR L=650 TO -650 STEP -4
30 SOUND 780-ABS(L),.3
40 L=L-2/650
50 NEXT L
60 GOTO 20

10 '-- Высокий и низкий гудки -------->
20 SOUND 987.7,5
30 SOUND 329.63,5
40 GOTO 20

10 '-- Звук работающего мотора ------->
20 FOR L=50 TO 60 STEP 10
30 SOUND L,.002
40 NEXT L
50 GOTO 20
```

Рис. 14.2. Примеры программ для создания некоторых звуковых эффектов.

делом затруднительным. В расширенном Бэйсике вместо оператора SOUND можно использовать оператор PLAY, который предусматривает применение специального музыкального языка, облегчающего программирование мелодий. Такой язык состоит из 19 подкоманд, перечисленных в табл. 14.2. Чтобы исполнить какую-нибудь мелодию, надо соответствующим образом выбрать номер строки, содержащей нужную последовательность подкоманд.

Обозначение нот

Существуют два способа указания нот в строке подкоманды оператора PLAY. Можно обозначать ноты буквами, например

```
100 PLAY "C D E F G A B"
```

Таблица 14.2. **Подкоманды оператора PLAY**

Подкоманда	Назначение подкоманды
нота	Звучание названной ноты (C, D, E, F, G, A или B) в нужной октаве с диезом (знак + или #), с бемолем (знак —) либо чистой (только буква)
O *октава*	Задание номера октавы от 0 до 6 (среднее C находится в октаве 3)
N *номер*	Звучание ноты с указанным номером от 0 до 84 (0 означает паузу)
L *длительность*	Задание длительности всех последующих нот, от целой ноты (*длительность* = 1) до шестьдесятчетвертой (*длительность* = 64); как вариант этот параметр может следовать за конкретной нотой, указывая именно ее длительность
P *длительность*	Пауза; параметр длительности задается как и в подкоманде L
точка	Эта подкоманда увеличивает длительность ноты или паузы в полтора раза
T *число ударов*	Задание темпа в пределах от 32 до 255 ударов в 1 мин
MF	Исполняется музыка, программа переводится в состояние ожидания
MB	Музыка исполняется на фоне продолжающейся программы
MN	Обычное исполнение (так называемое нон легато), не легато и не стаккато
ML	Исполнение легато
MS	Исполнение стаккато
XsS	Выполнение подкоманд очередной строки

В данном случае исполняется полная (7-звуковая) диктоническая гамма, начинающаяся с «до» второй октавы.

Для повышения звука на полтона надо рядом с его обозначением, нотой, поставить знак # или «+», а для понижения поставить знак «—». Диезы и бемоли, не имеющие соответствующих черных клавиш на фортепьяно, использовать не разрешается, т. е. недопустимы сочетания B-диез, E-диез, C-бемоль, F-бемоль.

Для перехода из одной октавы в другую используют подкоманду O. С помощью оператора PLAY можно исполнять мелодии в семи октавах, из которых три лежат ниже первой (номера с 0 по 2), а четыре — выше (номера с 3 по 6).

Пример. Воспроизведение всего звукоряда, начинающегося с «до» первой октавы (включая полутона).

```
110 PLAY "O3 C C# D D# E F F# G G# A A# B".
```

Всего в семи используемых октавах содержится 84 звука (рис. 14.3). Вместо того чтобы определять их номером октавы, обозначением и наименованием, можно указать просто порядковый номер ноты, используя подкоманду N.

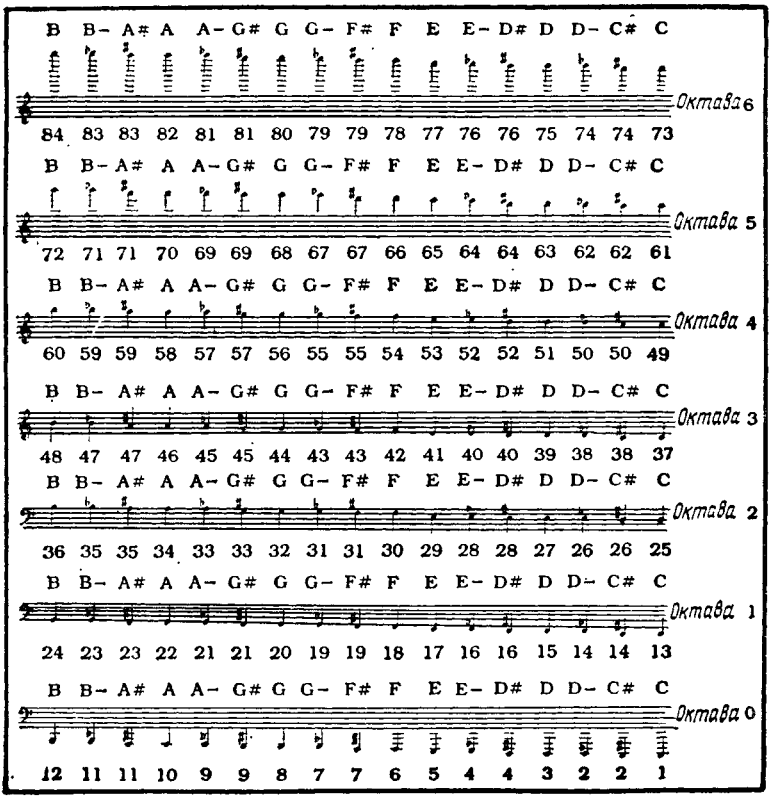


Рис. 14.3. Ноты, которые можно генерировать при помощи оператора PLAY (номера под нотами предназначены для использования в подкоманде N).

Пример. Исполнение той же гаммы, что и в предыдущем примере.

```
PLAY "N37 N38 N39 N40 N41 N42 N43 N44 N45 N46 N47 N48"
```

Длительность нот

До сих пор в приведенных примерах фигурировали лишь ноты, длительность которых составляла $^1/_4$ целой ноты. В общем случае для указания длительности ноты рядом с ее обозначением проставляют соответствующее число:

```
PLAY "03 C1 C2 C4 C8 C16 C32 C64"
```

Длительность воспроизводимой ноты равна единице, деленной на число, следующее за ее обозначением. Таким образом, число 1 указывает целую ноту, число 2 — половинную и т. д. В приведен-

ном примере звучат разные по длительности «до» первой октавы: целая, половинная, четвертная, восьмая, шестнадцатая, тридцать вторая и шестьдесят четвертая (рис. 14.4).

Можно объявить длительность ноты принимаемой по умолчанию, используя подкоманду L. Для этого после обозначения подкоманды следует поставить число, соответствующее желаемой длительности, как описано выше.

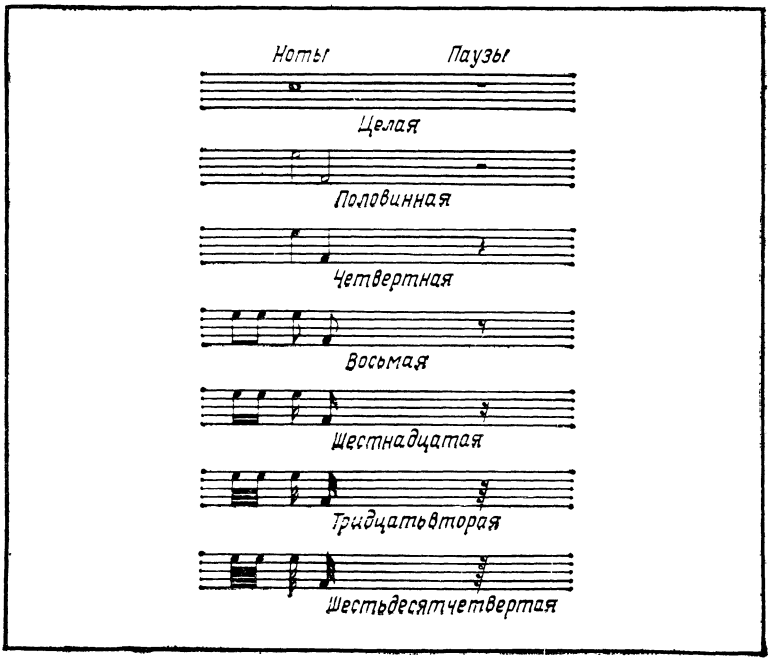


Рис. 14.4. Длительности нот и пауз.

Пример. Исполнение нисходящей гаммы, длительность звучания каждой ноты которой составляет $^1/_8$ целой ноты.

```
PLAY "02 L8 C B A G F E D"
```

Для удлинения ноты необходимо поставить точку после числа, указывающего ее длительность. Можно после ноты поставить и несколько точек; в этом случае каждая из них увеличивает длительность звука в полтора раза.

Пример. Исполнение удлиненных половинной и восьмой нот.

```
PLAY "C2. G8."
```

Паузы

С помощью подкоманды P устанавливаются паузы, длительность которых указывается соответствующим числом точно так же, как и при использовании подкоманды L (рис. 14.4).

Пример

```
PLAY "03 C P1 C P2 C P4 C P8 C P16 C"
```

Темп

Темп (или скорость) исполнения музыкального сочинения обозначается словами allegro, andante и др. или числом ударов метронома в 1 мин. Последний способ используется при установлении темпа с помощью подкоманды T. В табл. 14.1 приведены числа ударов метронома, соответствующие некоторым музыкальным темпам.

Музыкальные передний план и фон

Обычно в расширенном Бэйсике не предусматривается переход к следующей ноте мелодии до тех пор, пока не закончится звучание текущей. Такой способ исполнения музыкального произведения называется *музыкальным передним планом*. Однако посредством подкоманды MB расширенного Бэйсика возможно сообщить ПВМ инструкцию обработать до 32 нот, запомнить их в буферном запоминающем устройстве и затем исполнить эти ноты, одновременно продолжая выполнение программы. Такой режим называется *музыкальным фоном*; он может быть задан ПВМ, когда речь идет об исполнении коротких мелодий, содержащих не более 32 нот и пауз.

Приемы извлечения звука: staccato, legato, non legato

С помощью оператора PLAY звуки можно исполнять с четкими перерывами между ними — staccato, без перерывов — legato и с короткими перерывами — non legato, вполне достаточными для того, чтобы отличить звуки друг от друга. Исполнение staccato с помощью подкоманды MS достигается только за счет того, что выдерживается $^3/_4$ номинальной длительности каждого звука и $^1/_4$ оставляется на паузу. Для исполнения legato с помощью подкоманды ML выдерживается полная длительность каждого звука, в результате чего создается ощущение, что соседние звуки плавно переходят один в другой. Исполнение non legato обеспечивается подкомандой MN, при этом выдерживается $^7/_8$ номинальной длительности звука, а $^1/_8$ приходится на паузу.

Исполнение музыкальных произведений

С помощью оператора PLAY можно добиться вполне приемлемого звучания простых мелодий, в особенности музыкальных произведений, написанных для клавишных инструментов, таких, как фортепьяно или орган. Однако исполнение при этом ограничивается только одной партией. Проиллюстрируем сказанное на примере исполнения «Менуэта» и «Менуэта ре-минор» Иоганна Себастьяна Баха из «*Нотной тетради Анны-Магдалины*» (рис. 14.5 и 14.6). Программы для их исполнения приведены на рис. 14.7 и 14.8. Чтобы легче было сопоставлять программы с соответствующими нотными записями, в партитуре под каждой нотой напечатано ее буквенное название. Ниже дается краткий анализ работы программ и поясняется, каким образом партитуры обоих произведений были переведены на язык подкоманд.

Обе программы работают одинаково. Подкоманды оператора PLAY заносятся в операторы DATA, начиная со строки 1010; далее строки подкоманд считываются такт за тактом в последовательный массив TUNE$. После этого исполнение мелодии, хранящейся в массиве, становится просто вопросом использования подкоманды X, которая устанавливает очередность исполнения каждого элемента массива.

В партитуре «Менуэта» указан темп moderato (рис. 14.5), которому в табл. 14.1 соответствует диапазон 108—120 ударов в 1 мин. Подкомандой T в первом такте (рис. 14.7, строка 1010) устанавливается темп в 110 ударов в 1 мин. Подкомандой T объявляется длительность нот по умолчанию, равная $^1/_8$. Точка над первой нотой первого такта означает, что ее нужно играть staccato, поэтому далее следует подкоманда MS. Сама нота — это «ре» четвертой октавы (принимаемой по принципу умолчания) длительностью в $^1/_4$; поэтому ее исполнение реализуется подкомандой D4. Следующая подкоманда MN устанавливает прием исполнения non legato, поскольку над остальными нотами в этом такте нет точек. Очередная нота находится в третьей октаве, что и оговорено подкомандой O3. Подкомандами G, A и B инициируется исполнение соответствующих звуков длительностью в $^1/_8$. Последняя нота в этом такте — «до» четвертой октавы — реализуется подкомандами O4 и C. Каждый из оставшихся 31 такта этого сочинения аналогично преобразуется в один оператор DATA (строки с 1020 по 1320).

Темп второго произведения (рис. 14.6) — andante. Первой подкомандой (рис. 14.8, строка 1010) устанавливается темп, равный 90 ударам в 1 мин; эта величина находится в диапазоне, указанном в табл. 14.1. Затем в соответствии с партитурой подкомандой ML задается прием исполнения legato. Остальные операторы DATA непосредственно преобразуют партитуру в подкоманды.

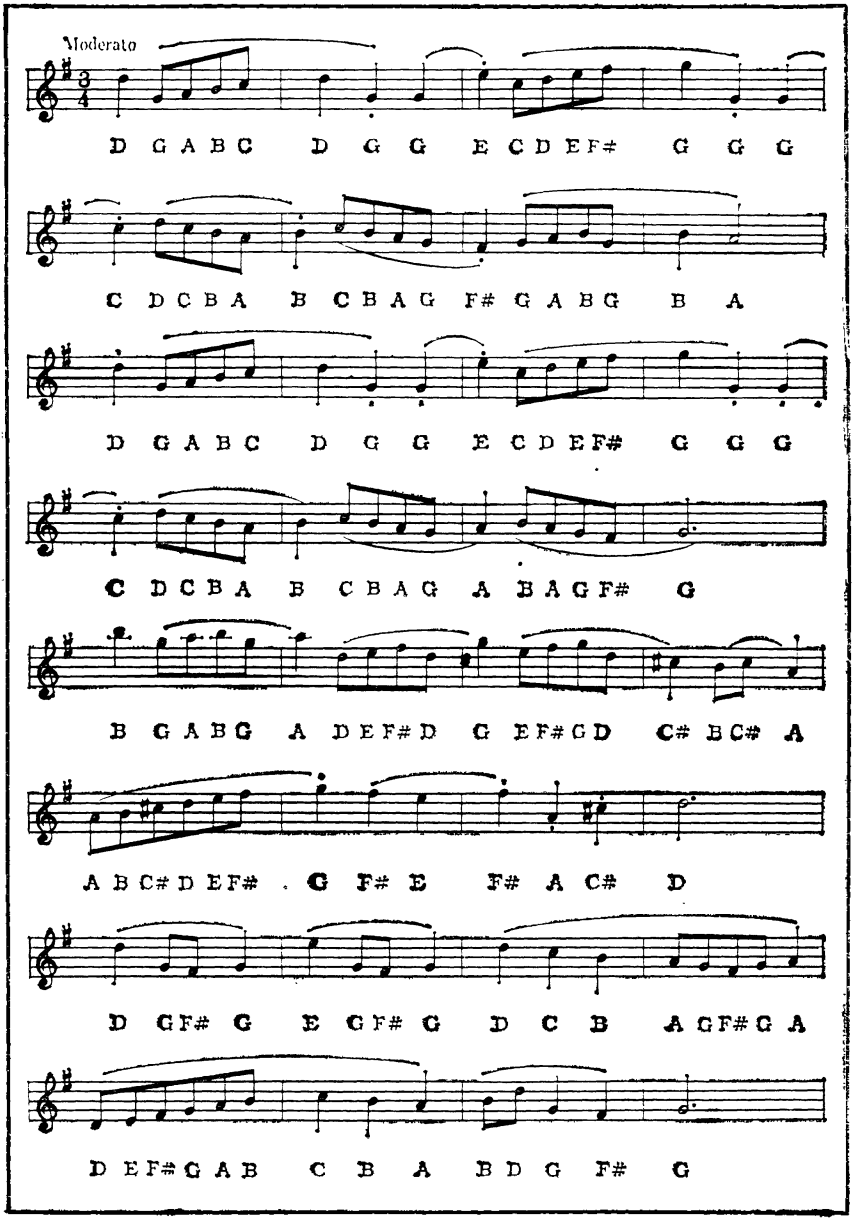


Рис. 14.5. «Менуэт» из «Нотной тетради Анны-Магдалины» И. С. Баха.

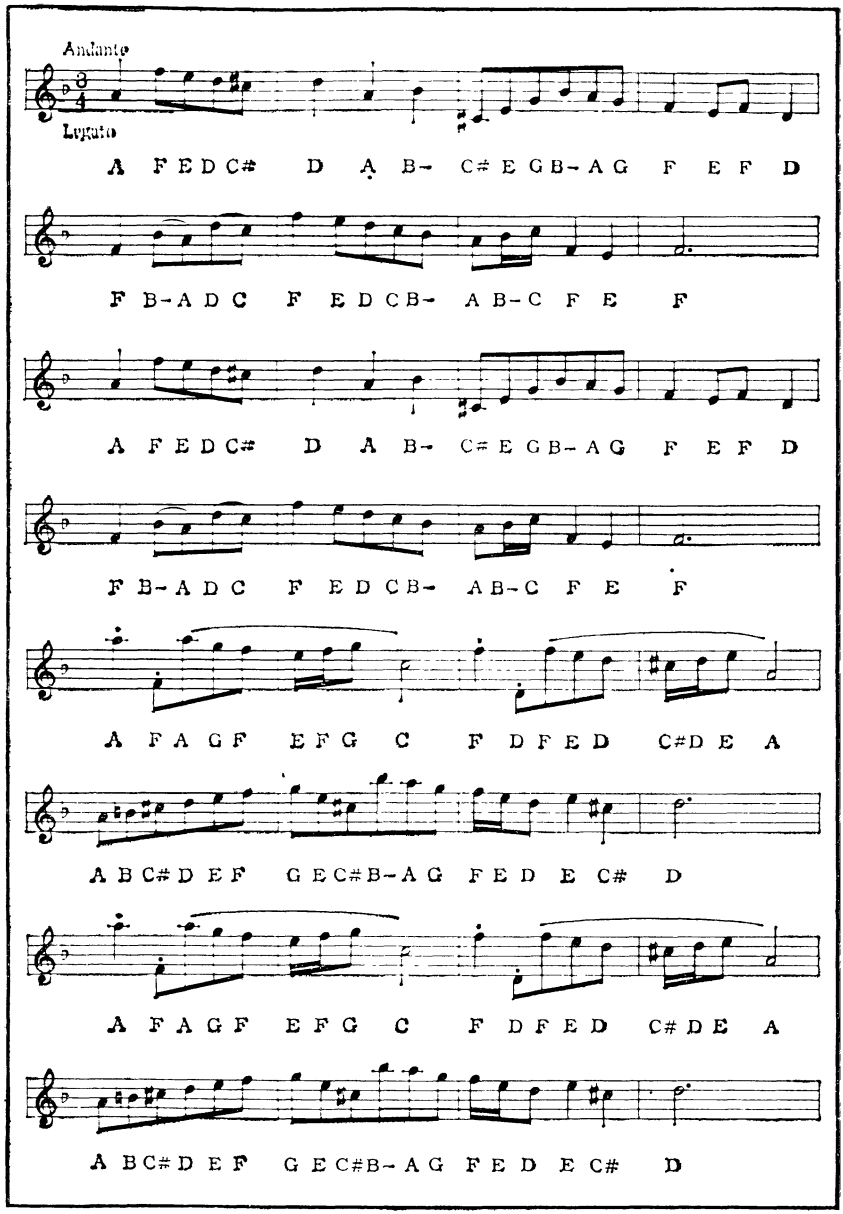


Рис. 14.6. «Менуэт ре-минор» из «Нотной тетради Анны-Магдалины» И. С. Баха.

```
10 DIM TUNE$(4)
20 CLS:WIDTH 40:KEY OFF
30 LOCATE 11,12:PRINT "Менуэт ре-минор"
40 LOCATE 12,17:PRINT "И.С.Бах "
90 REM '--Считывание музыкального произведения из массива данных--
100 FOR MEASURE=1 TO 32
110 READ NOTES$
120 TUNE$(MEASURE\8)=TUNE$(MEASURE\8)+NOTES$
130 NEXT MEASURE
190 '--Исполнение мелодии----------------------------
200 PLAY "xTUNE$(0); xTUNE$(1); xTUNE$(2); xTUNE$(3); xTUNE$(4);"
1000 '--Воспроизведение мелодии такт за тактом-----------
1010 DATA T90 ML L8 O3 A4 O4 F E D C#
1020 DATA D4 O3 A4 B-4
1030 DATA O3 C# E G B- A G
1040 DATA F4 E F D4
1050 DATA F4 B- A O4 D C
1060 DATA F4 E D C O3 B-
1070 DATA A B-16 O4 C16 O3 F4 E4
1080 DATA F2.
1090 DATA A4 O4 F E D C#
1100 DATA D4 O3 A4 B-4
1110 DATA C# E G B- A G
1120 DATA F4 E F D4
1130 DATA F4 B- A O4 D C
1140 DATA F4 E D C O3 B-
1150 DATA A B-16 O4 C16 O3 F4 E4
1160 DATA F2.
1170 DATA MS O4 A4 O3 F ML O4 A G F
1180 DATA E16 F16 G C2
1190 DATA MS F4 O3 D ML O4 F E D
1200 DATA C#16 D16 E O3 A2
1210 DATA A B O4 C# D E F
1220 DATA G E C# B- A G
1230 DATA F16 E16 D E4 C#4
1240 DATA D2.
1250 DATA MS A4 O3 F ML O4 A G F
1260 DATA E16 F16 G C2
1270 DATA MS F4 O3 D ML O4 F E D
1280 DATA C#16 D16 E O3 A2
1290 DATA A B O4 C# D E F
1300 DATA G E C# B- A G
1310 DATA F16 E16 D E4 C#4
1320 DATA D2.
```

Рис. 14.7. Программа для машинного исполнения «Менуэта» И. С. Баха (рис. 14.5).

```
10 DIM TUNE$(4)
20 CLS:WIDTH 40:KEY OFF
30 LOCATE 11,17:PRINT "Менуэт"
40 LOCATE 12,15:PRINT "И.С. Бах"
90 REM '-- Считывание музыкального произведения из массива данных --
100 FOR MEASURE=1 TO 32
110 READ NOTES$
120 TUNE$(MEASURE\8)=TUNE$(MEASURE\8)+NOTES$
130 NEXT MEASURE
190 '--Исполнение мелодии ----------
200 PLAY "xTUNE$(0); xTUNE$(1); xTUNE$(2); xTUNE$(3); xTUNE$(4);"
1000 '-- Воспроизведение мелодии такт за тактом ----------
1010 DATA T110 L8 MS D4 MN O3 G A B O4 C
1020 DATA D4 O3 MS G4 MN G4
1030 DATA O4 MS E4 MN C D E F#
1040 DATA G4 O3 MS G4 MN G4
1050 DATA MS O4 C4 MN D C O3 B A
1060 DATA MS B4 MN O4 C O3 B A G
1070 DATA MS F#4 MN G A B G
1080 DATA B4 A2
1090 DATA O4 MS D4 MN O3 G A B O4 C
1100 DATA D4 O3 MS G4 G4
1110 DATA O4 E4 MN C D E F#
1120 DATA G4 O3 MS G4 G4
1130 DATA O4 C4 MN D C O3 B A
1140 DATA B4 O4 C O3 B A G
1150 DATA A4 B A G F#
1160 DATA G2.
1170 DATA O4 B4 G A B G
1180 DATA A4 D E F# D
1190 DATA G4 E F# G D
1200 DATA C#4 O3 B O4 C# O3 A4
1210 DATA A B O4 C# D E F#
1220 DATA MS G4 MN F#4 E4
1230 DATA MS F#4 O3 A4 O4 C#4
1240 DATA MN D2.
1250 DATA D4 O3 G F# G4
1260 DATA O4 E4 O3 G F# G4
1270 DATA O4 D4 C4 O3 B4
1280 DATA A G F# G A4
1290 DATA D E F# G A B
1300 DATA O4 C4 O3 B4 A4
1310 DATA B O4 D O3 G4 F#4
1320 DATA G2.
```

Рис. 14.8. Программа для машинного исполнения «Менуэта ре-минор» И. С. Баха (рис. 14.6).

Глава 15

ПРЯМОЙ ДОСТУП И УПРАВЛЕНИЕ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ

По сравнению с Бэйсиком для других машин Бэйсик ПВМ значительно более совершенен, так как обеспечивает доступ к большинству ресурсов вычислительной системы. Однако с помощью операторов или функций Бэйсика нельзя управлять некоторыми средствами и дополнительными опциями ПВМ, в том числе заданием функции ключа Num Lock или переключением адаптера монохроматических и цветных графических устройств при наличии обоих типов. В ряде ситуаций Бэйсик не обеспечивает требуемого быстродействия при управлении предусмотренными средствами ПВМ, как, например, в случае «оживления» изображений на экране дисплея. Кроме того, системные программные средства расширенного и дискового Бэйсика позволяют использовать только около 92 000 байт динамической памяти, а кассетный Бэйсик — всего лишь 65 000 байт, несмотря на то что память системы может иметь гораздо большую емкость [1].

Для преодоления указанных ограничений в Бэйсике ПВМ предусмотрено несколько операторов и функций, обеспечивающих прямой доступ к памяти системы. В настоящей главе поясняется их использование и рассматривается ряд практических примеров.

Операторы и функции, объясняемые в данной главе, требуют осторожного обращения, поскольку все они обходят стандартные средства защиты Бэйсика и позволяют, следовательно, менять любое содержимое динамической памяти, включая Бэйсик-интерпретатор, ДОС-ПВМ, а также программы и переменные Бэйсика. Внеся изменения не в ту область памяти, можно создать ситуацию, при которой придется возвращать систему в исходное состояние с помощью клавиш Ctrl|Alt|Del. Возможно даже, что при этом потребуется полностью выключить систему, выждать несколько секунд и включить ее снова для восстановления функций управления.

Прямое обращение к памяти

Использование памяти ПВМ организуется самым различным образом. Для выполнения даже элементарной Бэйсик-программы система должна выделить ей сегмент памяти для хранения программных строк и сегмент для записи переменных Бэйсика. Несколько сегментов требуется для размещения Бэйсик-интерпретатора, его рабочей области, информации, выдаваемой на экран дисплея, и рабочего стека, не говоря уже об управляющей программе дисковой операционной системы.

Каждая ячейка памяти содержит число между 0 и 255, которое может представлять собой код символа, команду Бэйсика, код состояния некоторой части оборудования ПВМ или команду машинного языка. Если используется прямое обращение к памяти, то

[1] Программы на ассемблере не подвержены такому ограничению, поэтому в системах с памятью большой емкости их можно делать значительно более длинными, чем Бэйсик-программы.

именно программист должен определить, каким образом следует интерпретировать число, хранящееся в конкретной ячейке памяти.

В большинстве случаев программистам, работающим с Бэйсиком, нет необходимости задумываться над тем, каким образом устроена память ПВМ или какой смысл имеет то или иное число в отдельной ее ячейке. Это требуется лишь тогда, когда некоторыми возможностями или средствами нельзя управлять иначе как с помощью операторов DEF SEG, POKE и функции PEEK, которые открывают доступ к отдельным ячейкам памяти. Предусмотрена также команда CLEAR, позволяющая управлять размером программного стека.

Адресация памяти

Каждая отдельная ячейка памяти ПВМ определяется номером, который называется ее *адресом*. Персональная ЭВМ может иметь до 1 048 576 ячеек памяти (1024К байт) с адресами от 0 до 1048575. Часть этой области адресов относится к динамической памяти, часть — к ПЗУ, а некоторая часть ячеек в большинстве ПВМ остается вообще неиспользуемой. Это бывает в системах с объемом памяти меньше максимально допустимого. В силу специфики работы применяемого в ПВМ микропроцессора 8088 нельзя задавать нужный адрес памяти ПВМ простым указанием номера между 0 и 1048575. Сначала должен быть определен базовый адрес, называемый *адресом сегмента*, а затем указано смещение относительно базового адреса. При этом для вычисления действительного (абсолютного) адреса ПВМ умножает адрес сегмента на 16 и прибавляет смещение.

Пример

Если десятичный адрес сегмента 64 (в шестнадцатеричной системе это число 40), а смещение 23 (17 — в шестнадцатеричной системе), то действительный адрес будет 64*16+23=1047 (шестнадцатеричное число 417).

На величину адреса сегмента или смещения не накладывается никаких ограничений, за исключением того, что они должны лежать в пределах от 0 до 65535. Существует, следовательно, несколько различных комбинаций адреса сегмента и смещения, дающих один и тот же абсолютный адрес.

Пример

Если адрес сегмента 0, а смещение равно 1047 (417_{16}), то действительный адрес будет равным 0*16+1047=1047 (417_{16}), т. е. точно таким же, как и вычисленный абзацем выше.

Адреса памяти ПВМ, как правило, записываются в форме *сегмент:смещение*, а адрес сегмента и смещение обычно указываются в шестнадцатеричной системе.

Пример
Адрес 1047 (417_{16}) можно записать как 0:417 или 40:17.

Существует ряд адресов со смещением, отсчитываемым от начала рабочей области Бэйсика, адрес которой меняется в зависимости от используемой версии этого языка. Такие адреса записываются в форме DS:*смещение*.

Определение адреса сегмента памяти

Оператор DEF SEG задает адрес сегмента памяти, точнее, сегментную часть адреса, имеющего вид *сегмент:смещение*. Для указания адреса сегмента используется шестнадцатеричное число, как, например, в строке программы

```
100 DEF SEG=&H40
```

В системе указанная величина умножается на 16, а результат используется в качестве базового адреса для всех последующих операторов прямого доступа к памяти и функций типа PEEK, POKE, BLOAD, BSAVE, CALL и DEF USR. Значение адреса сегмента должно находиться в пределах от 0 до 65535.

Часть оператора DEF SEG, определяющая адрес сегмента памяти, является факультативной. Если она отсутствует, то в качестве базового адреса автоматически принимается базовый адрес рабочей области Бэйсика ПВМ. В этой форме оператор DEF SEG определяет первую часть конструкции DS:*адрес сегмента* и выглядит следующим образом:

```
1310 DEF SEG 'Используйте адрес сегмента рабочей области
     Бэйсика
```

Когда Бэйсик-интерпретатору впервые передается управление системой, в качестве адреса сегмента принимается базовый адрес рабочей области Бэйсика. В дальнейшем оператор DEF SEG может изменить этот адрес сегмента на любой другой. Оператор RUN, как и другие операторы, не возвращает адрес сегмента в рабочее поле Бэйсика; это может осуществиться только при повторном запуске интерпретатора или с помощью обычного оператора DEF SEG.

Проверка состояния и изменение содержимого памяти

Проверить содержимое любой ячейки памяти позволяет функция PEEK. Однако следует помнить, что это функция, а не оператор, поэтому надо использовать ее соответствующим образом.

Пример

```
200 DEF SEG=0:LW=PEEK(&H4A) 'Установите длину строки
```

Значение в скобках определяет смещение для текущего сегмента, которое должно находиться в пределах от 0 до 65535 (в шестнадцатеричной системе FFFF); оно определяет поле *смещения* в структуре адреса *сегмент:смещение*.

Функция PEEK сообщает численное значение в интервале от 0 до 255, содержащееся в проверяемой ячейке памяти. Эту величину можно интерпретировать как код символа, команду Бэйсика, номер строки, часть числа, состоящего из нескольких байт, и т. д. В некоторых случаях правильную интерпретацию можно дать по контексту, однако, если должной уверенности нет, надо анализировать те ячейки, использование которых известно.

Содержимое любой ячейки динамической памяти можно изменить с помощью оператора POKE, однако содержимое ПЗУ и адреса свободной области оперативной памяти таким способом изменить нельзя.

Пример

```
225 DEF SEG=0:POKE &H41A,PEEK(&H41C)
    'Сброс клавиатуры
```

Первая величина в операторе POKE, значение которой должно лежать между 0 и 65535 (в шестнадцатеричной системе FFFF), определяет смещение текущего сегмента. Эта часть определяет поле смещения в структуре адреса *сегмент:смещение*. Значение второй величины помещается в указанную ячейку памяти; оно должно лежать между 0 и 255 (в шестнадцатеричной системе FF).

Несколько полезных для запоминания адресов

В табл. 15.1 приведено несколько адресов памяти, используемых в операторе POKE и в функции PEEK; некоторые из них описаны более подробно ниже, при этом рассматриваются возможные пути обхода ряда «неудобных» особенностей, присущих ПВМ как вычислительной системе.

Изменение цвета символов на экране дисплея средней разрешающей способности

Текст и другие символы отображаются на экране дисплея среднего разрешения обычно в цвете, обозначаемом кодом 3, и могут быть белыми или золотистыми в зависимости от используемой палитры (табл. 13.3).

Цвет символов на экране изменяется после того, как выполнен составной оператор вида

```
1300 DEF SEG:POKE &H4E,HUE
```

Таблица 15.1. **Некоторые важные адреса памяти**

Название ячейки	Величина смещения в относительных адресах		
	шестнадцатеричное представление	десятичное представление	число байт
Для команды вида DEF SEG			
Текущий номер строки Бэйсик-программы	&H02E	46	2 [1]
Текущий номер строки с ошибкой	&H347	839	2 [1]
Смещение для начала текста Бэйсик-программы	&H030	48	2 [1]
Смещение для начала области переменных Бэйсика	&H358	856	2 [1]
Управление буфером клавиатуры для Бэйсика [2]	&H06A	106	1
Выбор цвета текстовых символов на экране дисплея среднего разрешения [2]	&H04E	78	1
Для команды вида DEF SEG=0			
Выбор адаптерной платы [2]	&H410	1040	1
Режим работы клавиатуры [2]	&H417	1047	1
Указатель начала системного буфера клавиатуры [2]	&H41A	1050	2
Указатель конца системного буфера клавиатуры	&H41C	1052	2
Системный буфер клавиатуры	&H41E	1054	16
Текущий режим экрана	&H449	1097	1
Ширина экрана (число колонок)	&H44A	1098	2

[1]) Для записи величин, которые могут быть больше 255, как, например, номер строки, требуются две соседние ячейки памяти. В этом случае значение величины равно сумме числа в первой ячейке и числа во второй ячейке, умноженного на 256.
[2]) Пояснения см. в тексте.

В данном примере переменная HUE должна принимать значения 1, 2 или 3; каждое значение соответствует одному из основных используемых цветов. Когда переменная HUE принимает нулевое значение, символы на экране делаются по цвету одинаковыми с фоном и становятся невидимыми. При этом символы, набираемые на клавиатуре, не отображаются на экране, благодаря чему ввод с клавиатуры надежно блокируется.

Если видимость символов не будет восстановлена программным способом с помощью какого-либо другого оператора POKE (с одновременным восстановлением и возможностью ввода с клавиатуры), то команды нельзя вводить без повторного рестарта системы.

Выбор видеоадаптера и задание ширины экрана

Если в ПВМ имеются и адаптер монохроматического дисплея, и адаптер цветных графических устройств, то одновременное их использование невозможно. При включении или возобновлении работы ПВМ выбор нужного видеоадаптера определяется положением переключателей выбора конфигурации системы [разд. "Options" в томе документации IBM "Guide to Operations" («Руководство по эксплуатации»)]. Требуемый адаптер может быть выбран программно путем изменения содержимого ячейки памяти с относительным адресом 0:410.

Пример

```
10 '— — Переход на 40-символьный цветной монитор — — — — — —
20 DEF SEG=0:POKE &H410,(PEEK(&H410) AND
   &HCF) OR &H10
30 SCREEN 1:SCREEN 0:WIDTH 40 '40-символьная строка
40 LOCATE ,,1,6,7 'форма курсора
10 '— — Переход на 80-симв. монохр. монитор — — — —
20 DEF SEG=0:POKE &H410,PEEK(&H410) OR &H30
30 SCREEN 0:WIDTH 80 '80-символьная строка
40 LOCATE ,,1,12,13 'форма курсора
```

Содержимое той же ячейки памяти определяет, какой из адаптеров включен:

```
10'— — Проверка выбранного адаптера — — — — — — — — —
20 DEF SEG=0
30 IF (PEEK(&H410) AND &H30)<>&H30 THEN 100
40 PRINT "Эта программа требует использования адаптера цветного
   монитора и графического дисплея."
50 END
100 REM далее продолжается программа графического вывода
```

Можно составить программу, предусматривающую использование как 40-символьной, так и 80-символьной строки в зависимости от возможностей конкретного монитора. В этом случае ячейка с адресом 0:4A будет указывать выбранную на данный момент ширину экрана.

Пример

```
200 DEF SEG=0:LW=PEEK(&H4A)  'Задание ширины экрана
```

Задание режима работы клавиатуры

Просто по клавиатуре нельзя определить, задан ли клавишей **Caps Lock** режим прописных букв. Точно так же нельзя сказать, на какой режим настроена система клавишей **Num Lock**: на ввод чис-

ловой или нечисловой информации. Оба факта устанавливаются путем проверки содержимого ячейки с адресом 0:417.

Пример

```
01 DEF SEG=0
20 '—— Проверка клавиши Shift Lock ——————————
30 LOCATE 25,39:IF (PEEK(&H417) AND &H40)=&H40
   THEN PRINT CHR$(24); ELSE PRINT CHR$(25);
40 '—— Проверка клавиши Num Lock ———————————
50 LOCATE 25,38:IF (PEEK(&H417) AND &H20)=&H20
   THEN PRINT "#"; ELSE PRINT CHR$(219);
```

В этой программе символы, указывающие режимы работы клавиатуры, отображаются в 25-й строке экрана. Если в 39-й позиции строки 25 появляется знак ↑, это означает, что используются прописные буквы, в противном случае будет стоять знак ↓. При настройке на ввод числовой информации в 38-й позиции отображается знак #; если вводится нечисловая информация, в этой позиции появляется изображение квадрата.

Одна и та же ячейка памяти может быть использована для задания нужного режима клавиатуры.

Пример

```
10 DEF SEG=0
20 POKE &H417,(PEEK(&H417) OR &H40) 'верхний регистр
20 POKE &H417,(PEEK(&H417) AND &HBF) 'нижний регистр
30 POKE &H417,(PEEK(&H417) OR &H20) 'ввод чисел
30 POKE &H417,(PEEK(&H417) AND&HDF) 'нечисловой
   ввод
```

Для установления нужного режима работы клавиатуры можно по своему усмотрению выбрать для включения в программу одну из двух строк с номером 20 или одну из двух строк с номером 30. По желанию можно обеспечить вывод на экран дисплея строки символов, указывающих заданный режим клавиатуры, как показано в предыдущем примере.

Игнорирование случайного нажатия клавиш

В ПВМ предусмотрен 15-символьный буфер клавиатуры, что позволяет работать на клавиатуре с максимальной скоростью даже при переключении вычислительной системы, например, с приема набираемых данных на обращение к диску. Буфер клавиатуры может очищаться программным путем непосредственно перед очередным запросом ввода с клавиатуры, благодаря чему надежно игнорируются все случайные нажатия клавиш, имевшие место после обслуживания предыдущего запроса ввода. Для достижения такого эффекта содер-

жимое ячейки с адресом 0:41A делается таким же, что и ячейки 0:41C.

Пример (рис. 11.4)

```
225 DEE SEG=0:POKE &H41A,PEEK(&H41C) 'Сброс
    клавиатуры
```

В Бэйсике ПВМ предусматривается собственный буфер клавиатуры, в котором накапливаются символы, порождаемые нажатием одиночной клавиши. Это необходимо потому, что, например, в результате нажатия отдельной функциональной клавиши генерируется целая цепочка символов. Буфер клавиатуры можно очистить от лишних символов с помощью следующей программной строки:

```
235 DEF SEG:POKE &H6A,0 'Очистка буфера
    клавиатуры в Бэйсике
```

Увеличение рабочего стека

Для запоминания вложенных циклов FOR/NEXT, вложенных вызовов программ (включая рекурсивные) и сложных операторов PAINT в Бэйсике ПВМ используется программный стек. Если глубина таких вложений слишком велика или программа «рисует» очень сложную цветную фигуру, программный стек может оказаться переполненным, и тогда появится сообщение об ошибке "Out of memory" («Нехватка памяти»). Эту ошибку можно предотвратить, увеличив размер стека с помощью оператора CLEAR

Пример

```
10 CLEAR ,,2048 'Задание размера стека
```

Указываемая пользователем величина определяет количество ячеек памяти, резервируемых под программный стек. Стандартный размер стека — 512 байт или 1/8 объема динамической памяти, доступного для Бэйсик-программ; эта величина устанавливается в тех случаях, когда пользователь указывает меньшую. Кроме того, оператор CLEAR стирает все переменные и массивы, оставляя неизменными строки программы (гл. 8).

Программирование на машинном языке

Предпочитая машинному языку Бэйсик, мы несколько проигрываем в быстродействии программ, но зато получаем возможность пользоваться более удобным и более естественным проблемно-ориентированным языком программирования. Для большинства задач прикладного программирования такое небольшое уменьшение скорости выполнения программ оказывается незаметным; однако в задачах типа машинной мультипликации скорость вычислений может

быть критичной, и тогда быстродействие Бэйсик-программы становится недостаточным. Бэйсик ПВМ обеспечивает наилучший выход из этого положения: можно применять Бэйсик в тех частях программы, где скорость вычислений несущественна, а с помощью оператора CALL или функции USR в программе на Бэйсике переходить к блокам программы на машинном языке в тех местах, где требуется максимальное быстродействие.

Никто не пишет программы для ПВМ прямо на машинном языке: для этого используется ассемблер, который является символической версией машинного языка. Команды ассемблера состоят из простых для запоминания буквенных комбинаций, каждая из которых может быть преобразована точно в одну команду машинного языка.

Для того чтобы квалифицированно программировать на ассемблере, необходимо знать, как использовать некоторые сервисные программы. К числу сервисных программ относятся:

- *редактор* вводимых программ;
- *ассемблер* для перевода исходной программы на машинный язык;
- *редактор связей*, создающий отображение программы на машинном языке в памяти машины;
- *отладчик*, служащий вспомогательным средством для проверки правильности программ и поиска ошибок в них.

Очевидно, что программирование на ассемблере — это сложная проблема, требующая длительного изучения и не являющаяся предметом рассмотрения данной книги. Для получения дополнительной информации по этому вопросу можно обратиться к следующим литературным источникам.

- IBM Personal Computer Disk Operating System, 2nd ed., part number 6024001; в главах 4—6 описываются способы использования текстового редактора, редактора связей и отладчика;
- IBM Personal Computer Basic, 2nd ed., part number 6025010; в приложении C объясняется, как включать в программу на Бэйсике блоки, написанные на машинном языке;
- IBM Personal Computer Macro Assembler, part number 6024002; показано, как пользоваться программами на языке ассемблера IBM;
- Rector, Russell and George Alexy, The 8086 Book: includes the 8088, Osborne/McGram-Hill, Berkeley, CA, 1980; в главах 1—6 книги обсуждается программирование на языке ассемблера для базового микропроцессора ПВМ — Intel 8088;
- Willin, David, 8088 Assembly Language Programming: The IBM PC, Howard W. Sams & Co., Indianapolis, Indiana, 1983; рассматриваются вопросы программирования ПВМ с использованием языка ассемблера.

Приложение А

КРАТКОЕ ОПИСАНИЕ БЭЙСИКА

В данном приложении содержится краткое описание всех команд, операторов и стандартных функций Бэйсика, а также список математических функций, получаемых путем комбинирования стандартных функций. Приводимые здесь сведения могут быть использованы при наличии интерпретатора дискового или расширенного Бэйсика версий 1.05 и 1.10 либо интерпретатора кассетного Бэйсика версии 1.0. Для каждой команды, оператора и функции приводятся сведения о том, какому варианту Бэйсика они принадлежат — кассетному, дисковому, расширенному или всем трем. Более подробную информацию о перечисленных здесь командах, операторах и функциях можно получить, пользуясь предметным указателем данной книги, либо обратившись к гл. 4 книги "IBM Personal Computer BASIC", 2nd ed., которая содержит описание Бэйсика ПВМ фирмы IBM.

При вводе рассматриваемых ниже команд с клавиатуры следует набирать все символы, выделенные ЖИРНЫМ ШРИФТОМ, в точности так, как они напечатаны, и только заглавные буквы разрешается в любом месте заменять на строчные и наоборот. Все напечатанное *курсивом* необходимо заменять на соответствующие фактические слова, константы, переменные, элементы массивов, выражения и т. п. Текст, напечатанный в квадратных скобках, является необязательным, однако если он используется в команде, операторе или функции, то квадратные скобки не набираются. Любой параметр, за которым следует многоточие (. . .), можно повторять многократно, причем само многоточие набирать не следует. Необходимо внимательно следить за тем, чтобы были проставлены все указанные знаки препинания (кроме квадратных скобок и многоточия), включая запятые, двоеточия, косую черту, восклицательные знаки, точки, знаки равенства и знак номера.

Команды

Перечисленные ниже команды наиболее широко используются в режиме немедленной обработки, однако три из них — NAME, TRON и TROFF — в ряде случаев оказываются полезными и в программируемом режиме. Роль своеобразных команд играют и некоторые комбинации клавиш при их совместном нажатии:

● **Ctrl|Scroll Lock** — прерывает текущую операцию, оставляя ПВМ в режиме немедленной обработки;

● **Ctrl|Alt|Del** — завершает выполнение Бэйсик-программ и возвращает ПВМ в исходное состояние;
● ⇧|**PrtSc** — выводит текст, высвеченный на экране, на печатающее устройство (обозначаемое как LPT1:);
● **Ctrl|Num Lock** — переводит экран дисплея в состояние приостановки синтеза динамического изображения.

Ниже разъясняется смысл остальных команд Бэйсика.

AUTO [*номер первой строки*][,[*приращение*]]
Установка режима автоматической нумерации программных строк. Нажатие комбинации клавиш **Ctrl** | **Scroll Lock** возвращает машину в режим ручной нумерации программных строк. (Во всех разновидностях Бэйсика.)

BLOAD *имя файла* [,[*смещение*]]
Считывание из файла с заданным именем информации, относящейся к определенной области памяти, и занесение ее в память с начального адреса этой области либо по адресу, определяемому *смещением*, если оно задано в команде (см. команду DEF SEG). (Во всех разновидностях Бэйсика.)

BSAVE *имя файла, смещение, длина*
Запоминание в файле с заданным именем информации из некоторой области памяти, которая начинается с адреса, определяемого смещением (см. команду DEF SEG), и имеет заданную длину. (Во всех разновидностях Бэйсика.)

CLEAR [,[*память программ*][,*стековая память*]]
Присваивание нулевых значений всем числовым и строковым переменным и элементам массивов. Задавая указанные в команде факультативные параметры, можно получить информацию об имеющемся свободном объеме памяти программ (используемой для хранения программных строк, переменных и организации рабочей области интерпретатора), а также об объеме динамической памяти, зарезервированной для рабочего стека. (Во всех разновидностях Бэйсика.)

CONT
Продолжение выполнения программы после останова, начиная с очередного оператора. (Во всех разновидностях Бэйсика.)

DELETE *номер первой строки*[—*номер последней строки*]
Исключение строк программы с заданными номерами. (Во всех разновидностях Бэйсика.)

EDIT *номер строки*
Вывод на экран заданной программной строки для редактирования. (Во всех разновидностях Бэйсика.)

FILES [*имя файла*]

Вывод на экран из дискового справочника тех имен файлов, которые соответствуют заданному в команде имени (разрешается использовать родовое имя в качестве заданного). (В дисковом и расширенном Бэйсике.)

LIST [*номер первой строки*][—[*номер последней строки*]] [, *имя файла устройства*]

Вывод на экран всех или части программных строк из памяти; возможна запись их в файл или вывод на устройство с заданным именем. (Во всех разновидностях Бэйсика.)

LLIST [*номер первой строки*][— [*номер последней строки*]]

Вывод на печатающее устройство (с именем LPT1:) всех или части находящихся в памяти программных строк. (Во всех разновидностях Бэйсика.)

LOAD *имя файла*[,R]

Загрузка программы из файла с заданным именем (с одновременным стиранием программных строк, находившихся в памяти до этого момента) и последующее ее выполнение в случае необходимости. (Во всех разновидностях Бэйсика.)

MERGE *имя файла*

Объединение программных строк из файла с заданным именем с находящимися в данный момент в памяти. (Во всех разновидностях Бэйсика.)

NAME *старое имя* **AS** *новое имя*

Переименование дискового файла. (В дисковом и расширенном Бэйсике.)

NEW

Удаление из памяти всех программных строк, переменных и массивов. (Во всех разновидностях Бэйсика.)

RENUM [[*первый новый номер строки*][,[*первый старый номер строки*][,*приращение*]]

Перенумерация программных строк. (Во всех разновидностях Бэйсика.)

RESET

Закрытие всех открытых файлов и устройств. (В дисковом и расширенном Бэйсике.)

RUN [*номер строки*]

Выполнение находящейся в памяти программы; если задан номер, то программа выполняется, начиная с указанной строки. (Во всех разновидностях Бэйсика.)

RUN *имя файла*[,R]

Загрузка программы из файла с заданным именем и ее выполнение; все находившиеся ранее в памяти программные строки стираются. Если задан факультативный параметр ([,R]), то файлы остаются открытыми. (Во всех разновидностях Бэйсика.)

SAVE *имя файла* [,*опция*]

Запись программы, находящейся в памяти, в файл с заданным именем. Если не задано никакой опции, то при записи используется сжатый формат; опция A определяет кодирование символов в коде ASCII, опция P запрещает представление программных строк. (Во всех разновидностях Бэйсика.)

SYSTEM

Передача управления операционной системе ДОС ПВМ. (В дисковом и расширенном Бэйсике.)

TRON

Установка режима трассировки программы. (Во всех разновидностях Бэйсика.)

TROFF

Отмена режима трассировки программы. (Во всех разновидностях Бэйсика.)

Операторы

Перечисленные ниже операторы наиболее распространены в программируемом режиме, хотя все они, за исключением двух, DATA и DEF FN, могут использоваться и в режиме немедленной обработки.

BEEP

Включение встроенного звукового устройства. (Во всех разновидностях Бэйсика.)

CALL *числовая переменная* [(*переменная* [,*переменная*,]...)]

Передача управления объектной программе, находящейся в памяти по относительному адресу, задаваемому числовой переменной (см. оператор DEF SEG). Для каждой заданной в команде переменной (необязательные параметры) в программу на машинном языке с помощью микропроцессорного стека передается адрес памяти, по которому находится значение этой переменной. (Во всех разновидностях Бэйсика.)

CHAIN [**MERGE**] *имя файла* [,[*выражение*][,[**ALL**]]
[,**DELETE** *номер первой строки*, *номер последней строки*]]

Загрузка программы из файла с заданным именем и ее выполнение. Когда факультативный параметр MERGE не задан, все находящиеся в памяти программные строки стираются, в противном случае они объединяются с загружаемыми. Если в операторе задано *выра-*

жение (необязательный параметр), то его значение используется в качестве номера первой программной строки, с которой должно начинаться выполнение программы; в противном случае выполнение начинается со строки с наименьшим номером. Если факультативный параметр ALL отсутствует, то все переменные (кроме перечисленных в предыдущих операторах COMMON) стираются. Использование факультативной конструкции DELETE позволяет удалить из памяти заданный диапазон программных строк перед объединением текущей программы с новыми загружаемыми строками. (В дисковом и расширенном Бэйсике.)

CIRCLE *(столбец, строка), радиус* [,[*цвет*][,*начальная точка дуги, конечная точка дуги*][,*характеристическое отношение*]]]

Построение на экране окружности, эллипса или дуги заданного радиуса, который измеряется количеством столбцов, с центром в точке с заданными координатами (*столбец, строка*). С помощью необязательного параметра *цвет* можно задать цвет выводимой окружности, эллипса или дуги; в качестве значений этого параметра могут использоваться номера 0 или 1 в режиме высокого разрешения или от 0 до 3 в режиме среднего разрешения (выбор одного из четырех цветов активной палитры) (см. табл. 13.3). Если номер цвета не задан, то используется стандартный цвет переднего плана. В тех случаях, когда заданы факультативные параметры *начальная точка дуги*, *конечная точка дуги*, на экране вместо полной окружности строится дуга с указанными концевыми точками (для задания начальной и конечной точек должны использоваться единицы измерения длин дуг — радианы, от 0 до 6.2831). Если концевая точка задана отрицательным значением, то дополнительно проводится радиус из центра окружности в эту концевую точку. При задании факультативного параметра *характеристическое отношение* строится эллипс с отношением высоты к ширине, указанным в операторе (отношение 5/6 определяет окружность в режиме среднего разрешения, а 5/12 — в режиме высокого разрешения). (В расширенном Бэйсике.)

CLOSE[#][*номер файла/устройства*][,[#][*номер файла/устройства*]] ...

Закрытие файлов и устройств с заданными *номерами*. Если не задано ни одного номера, то закрываются все файлы и устройства. (Во всех разновидностях Бэйсика.)

CLS

Очистка экрана дисплея. (Во всех разновидностях Бэйсика.)

COLOR [*цвет переднего плана*][,[*цвет фона*][,*цвет границы*]]

Установка цвета *переднего плана*, фона и границы для экрана, работающего в режиме текстового вывода. Возможные цвета перечислены в табл. 13.3. (Во всех разновидностях Бэйсика.)

COLOR [*цвет фона/границы*][,*палитра*]

Установка одного из 16 возможных цветов фона и границы и выбор одной из двух *палитр* переднего плана для экрана, работающего в режиме среднего разрешения. Допустимые цвета и палитры приведены в табл. 13.2 и 13.3. (Во всех разновидностях Бэйсика.)

COM *(адаптер) действие*

В качестве значения параметра *действие* может быть задано OFF, ON или STOP. В зависимости от этого производится блокирование (OFF), разблокирование (ON) или приостановка (STOP) последовательной передачи данных к адаптеру 1 или 2 (см. ON COM-GOSUB). (В расширенном Бэйсике.)

COMMON *переменная*[,*переменная*]...

Задание имен переменных и массивов, значения которых должны сохраняться при выполнении последующего оператора CHAIN. После каждого задаваемого имени массива должны стоять круглые скобки ( ). (В дисковом и расширенном Бэйсике.)

DATA *константа*,[*константа*,]...

Добавление строковых и числовых констант к программному списку значений для операторов READ. (Во всех разновидностях Бэйсика.)

DATE$-*строковое значение*

Установка текущей даты для системного календаря. (В дисковом и расширенном Бэйсике.)

DEF FN *имя*[(*фиктивная переменная*[,*фиктивная переменная*...)] =*определение*

Присваивание имени строковой или числовой функции и задание ее определения в виде выражения. Любое задаваемое в качестве необязательного параметра имя фиктивной переменной может использоваться в выражении, определяющем функцию (параметр *определение*). Фиктивные переменные заменяются на фактические значения всякий раз при обращении к функции (см. функцию FN). (Во всех разновидностях Бэйсика.)

DEF SEG[=*адрес*]

Задание текущего адреса сегмента; этот адрес, умноженный на 16, в дальнейшем будет автоматически прибавляться к адресным смещениям для получения действительных адресов в операторах BLOAD, BSAVE, CALL и POKE, а также в функциях PEEK и USR. Если параметр *адрес* отсутствует, то в качестве адреса сегмента используется адрес свободного пространства Бэйсик-интерпретатора. (Во всех разновидностях Бэйсика.)

DEF *буква типа* [— *буква*][,*буква* [— *буква*]]...

Определение *типа* для переменных, имена которых начинаются с букв, входящих в один из перечисленных диапазонов. Параметр

тип должен иметь значения: INT — для определения целого типа, SNG — с обычной точностью, DBL — с двойной точностью, а T — для определения строковых переменных.(Во всех разновидностях Бэйсика.)

DEF USR [*номер*]=*смещение*

Указание адреса *смещения* для объектной подпрограммы с заданным номером, которая вызывается с помощью функции USR. В качестве *номера* разрешается использовать любую цифру от 0 до 9. (Во всех разновидностях Бэйсика.)

DIM *имя массива* (*индекс*[,*индекс*]...) [,*имя массива*(*индекс*[,*индекс*]...)]...

Распределение памяти для массивов. Задаваемое в качестве параметра *индекс* числовое выражение определяет максимальное значение соответствующего индекса массива (см. OPTION BASE). (Во всех разновидностях Бэйсика.)

DRAW *строка подкоманд*

Построение на экране геометрического рисунка, определяемого заданной строкой подкоманд. Возможные подкоманды перечислены в табл. 13.4. (В расширенном Бэйсике.)

END

Останов программы, закрытие всех файлов и устройств и возврат в режим немедленной обработки. (Во всех разновидностях Бэйсика.)

ERASE *имя массива* [,*имя массива*]...

Стирание массивов с заданными именами. (Во всех разновидностях Бэйсика.)

ERROR *код*

Формирование кода ошибки из заданного списка (см. ON ERROR GOTO). (Во всех разновидностях Бэйсика.)

FIELD [#]*номер файла,длина* **AS** *строковая переменная*[,*длина* **AS** *строковая переменная*]...

Определение полей файла с прямым доступом. (В дисковом и расширенном Бэйсике.)

FOR *переменная-счетчик цикла=начальное значение* **TO** *конечное значение*[**STEP** *значение приращения*]

Начало цикла FOR/NEXT. (Во всех разновидностях Бэйсика.)

GET [#] *номер файла* [,*номер записи*]

Считывание записи из файла с произвольным доступом. Если задан факультативный параметр *номер записи*, то считывается запись с заданным номером, в противном случае — запись с очередным номером. (В дисковом и расширенном Бэйсике.)

GET *(столбец1,строка1) — (столбец2,строка2),имя массива*

Запоминание цвета всех точек прямоугольной области, заданной координатами двух диаметрально противоположных углов. Номера цветов записываются в массив с заданным именем. (В расширенном Бэйсике.)

GOSUB *номер строки*

Передача управления в подпрограмму строке с заданным номером (см. RETURN). (Во всех разновидностях Бэйсика.)

GOTO *номер строки*

Передача управления строке с заданным номером. (Во всех разновидностях Бэйсика.)

IF *условие* **THEN** *оператор*[*:оператор*]... [**ELSE** *оператор* [*:оператор*]...]

Выполнение того или иного набора операторов в зависимости от истинностного значения заданного условия. Если *условие* удовлетворяется, то выполняется первый набор операторов (стоящий непосредственно после слова THEN), в противном случае — второй (стоящий после слова ELSE). Если условие не удовлетворяется и конструкция ELSE отсутствует, то осуществляется переход к очередной программной строке. (Во всех разновидностях Бэйсика.)

IF *условие* **THEN GOTO** *номер строки* [**ELSE** *оператор* [*:оператор*] ...]

Выполнение тех или иных операторов в зависимости от истинностного значения условия. Если *условие* истинно, то управление передается строке с заданным *номером*, в противном случае выполняется второй набор операторов (операторы, стоящие непосредственно после слова ELSE). Если *условие* ложно, а конструкция ELSE не задана, то управление передается следующей программной строке. (Во всех разновидностях Бэйсика.)

INPUT [;]["*запрос*"; ***или*** "*запрос*",] *переменная*[,*переменная*]...

Присваивание вводимых с клавиатуры значений всем поименованным переменным. В качестве наводящего сообщения (запроса на ввод данных с клавиатуры) на экран выводится вопросительный знак, а за ним, возможно, факультативный текст запроса. Когда перед списком имен переменных стоит запятая, вопросительный знак в наводящем сообщении не выводится; если список предваряется точкой с запятой, то сообщение заканчивается знаком вопроса. Факультативная точка с запятой после слова INPUT приводит к подавлению сигнала возврата каретки, который обычно формируется при нажатии клавиши ↵, завершающем ввод очередной записи. (Во всех разновидностях Бэйсика.)

INPUT#*номер файла,переменная*[,*переменная*]...

Присваивание значений, считываемых из указанного файла или

устройства, по очереди всем переменным с заданными именами. Символы возврата каретки, перехода на новую строку, запятые, а также проблемы между числовыми значениями интерпретируются при этом как разделители значений. (Во всех разновидностях Бэйсика.)

KEY *номер,строковое значение*

Активизация функциональной клавиши с заданным *номером* (от 1 до 10) как программируемой клавиши и задание в качестве ее определения указанного *строкового значения*. Если это строковое значение — нулевое, то функциональная клавиша перестает считаться программируемой. (Во всех разновидностях Бэйсика.)

KEY *действие*

Значением параметра *действие* должно быть ON, OFF или LIST. Если задано ON, то на 25-й строке экрана выводятся по шесть первых символов каждого определения программируемых клавиш; если OFF, то они стираются, а если LIST, то в основной области экрана выводятся полностью все определения, т. е. по 15 символов на каждое. (Во всех разновидностях Бэйсика.)

KEY *(числовое значение) действие*

Значением параметра *действие* должно быть OFF, ON или STOP. В зависимости от этого запрещается (OFF), разрешается (ON) или приостанавливается (STOP) слежение за функциональной клавишей или заданной клавишей управления курсором (*числовое значение*, определяющее номер этой клавиши, должно заключаться между 1 и 14). (В расширенном Бэйсике.)

KILL *имя файла*

Удаление дискового файла с заданным именем. (В дисковом и расширенном Бэйсике.)

[LET] *переменная=выражение*

Присваивание значения *выражения* заданной *переменной*. (Во всех разновидностях Бэйсика.)

LINE [(*столбец*1,*строка*1)—(*столбец*2,*строка*2) [,[*цвет*][,B[F]]]

Построение на экране дисплея линии или прямоугольника. При использовании факультативного параметра F прямоугольник можно закрасить стандартным цветом переднего плана (если параметр *цвет* не задан) либо цветом с номером, заданным в качестве значения параметра *цвет*. При задании цвета допустимы номера 0 или 1 в режиме графического вывода с высоким разрешением или от 8 до 3 в режиме среднего разрешения для выбора одного из четырех цветов активной палитры (см. табл. 13.3). (Во всех разновидностях Бэйсика.)

LINE INPUT[;][*"запрос"*;] *строковое значение*

Ввод всех символов, набираемых на клавиатуре до нажатия клавиши ↵, и последующее присваивание их заданной *строковой пере-*

менной. Вопросительный знак в наводящем сообщении (запросе на ввод данных с клавиатуры) присутствует лишь тогда, когда он в явном виде включен в значение необязательного параметра *запрос.* Точка с запятой после слова INPUT подавляет возврат каретки, который обычно происходит при нажатии клавиши ↵ в конце ввода очередной записи. (Во всех разновидностях Бэйсика.)

LINE INPUT # *номер файла,строковая переменная*

Ввод всех символов из указанного файла вплоть до ближайшего возврата каретки, за которым следует символ перехода на следующую строку; все введенные символы присваиваются заданной *строковой переменной.* (Во всех разновидностях Бэйсика.)

LOCATE [*строка*][,*столбец*][,[*видимость*][,[*первый*][,*последний*]]]

Перемещение курсора в позицию, определяемую заданными номерами строки и столбца (используется нумерация строк и столбцов экрана, принятая в режиме текстового вывода). Значение параметра *видимость* определяет, должен ли курсор быть видимым на экране (1) или нет (0). Значения параметров *первый* и *последний* определяют размер и форму курсора. (Во всех разновидностях Бэйсика.)

LPRINT [**USING** *шаблон*;] *список значений*[;]

Печать на основном системном печатающем устройстве (LPT1:); в остальном производятся те же действия, что и в случае оператора PRINT. (Во всех разновидностях Бэйсика.)

LSET *переменная поля=строковое значение*

Присваивание строкового значения заданной переменной поля в буфере файла с произвольным доступом. Поле заполняется начиная с крайней левой позиции, оставшиеся неиспользованными правые позиции заполняются пробелами. Производится также выравнивание по крайней левой позиции значения стандартной строковой переменной. (В дисковом и расширенном Бэйсике.)

MID$(*строковая переменная,первый символ*[,*количество символов*])=*строковое значение*

Замена подстроки значения заданной *строковой переменной* на указанное справа строковое значение. Заменяемая подстрока начинается с заданного *первого символа.* С помощью факультативного параметра *количество символов* можно ограничить число заменяемых знаков. (Во всех разновидностях Бэйсика.)

MOTOR *состояние*

Пуск (состояние<0 или >0) или останов (состояние=0) ленты накопителя. (Во всех разновидностях Бэйсика.)

NEXT [*переменная-счетчик цикла*][,*переменная-счетчик цикла*]...

Окончание одного или более FOR/NEXT-циклов. (Во всех разновидностях Бэйсика.)

ON COM *(адаптер)* **GOSUB** *номер строки*

Задание *номера* подпрограммной строки, которой должно передаваться управление в случае обнаружения факта последовательной передачи данных к адаптеру 1 или 2 (см. COM-ON). (В расширенном Бэйсике.)

ON ERROR GOTO *номер строки*

Задание номера программной строки, которой должно передаваться управление в случае выявления ошибки (см. RESUME). (Во всех разновидностях Бэйсика.)

ON *выражение* **GOSUB** *номер строки*[,*номер строки*]...

Передача управления в подпрограмму строке с одним из указанных номеров в зависимости от конкретного значения выражения (см. RETURN). (Во всех разновидностях Бэйсика.)

ON *выражение* **GOTO** *номер строки*[,*номер строки*]...

Переход к строке с одним из указанных *номеров* в зависимости от конкретного значения заданного *выражения*. (Во всех разновидностях Бэйсика.)

ON KEY *(числовое значение)* **GOSUB** *номер строки*

Задание *номера* подпрограммной строки, которой должно передаваться управление при нажатии функциональной клавиши или клавиши управления курсором, номер которой определяется заданным числовым значением (от 1 до 14) (см. KEY-ON). (В расширенном Бэйсике.)

ON PEN GOSUB *номер строки*

Задание *номера* подпрограммной строки, которой должно передаваться управление, если обнаруживается, что световое перо находится в активном состоянии (см. PEN ON). (В расширенном Бэйсике.)

ON STRIG *(числовое значение)* **GOSUB** *номер строки*

Задание номера подпрограммной строки, которой должно передаваться управление, если обнаруживается, что триггер управления электронной игрой, номер которого определяется заданным *числовым значением* (0, 2, 4 или 6), находится в активном состоянии (см. STRIG ON). (В расширенном Бэйсике.)

OPEN *имя файла/устройства* [**FOR** *режим*] **AS**[#]*номер файла/устройства* [**LEN**=*длина записи*]

Присваивание *файлу* или *устройству* с заданным именем указанного номера и установление *режима доступа*, который будет использоваться применительно к этому файлу. Если в качестве значения параметра *режим* задано INPUT, OUTPUT или APPEND, то устанавливается режим последовательного доступа. Если конструкция FOR отсутствует, то устанавливается режим произвольного до-

ступа. С помощью соответствующего факультативного параметра можно задать длину записи в режиме произвольного доступа. (Во всех разновидностях Бэйсика.)

OPEN *режим* [#] *номер файла/устройства, имя файла/устройства* [,*длина записи*]

Видоизмененная форма предыдущего оператора OPEN. В данном случае способ доступа определяется значением параметра *режим* следующим образом: при значении, равном 0, организуется последовательный вывод, при значении 1 — последовательный ввод, а при значении R — произвольный доступ. (Во всех разновидностях Бэйсика.)

OPEN "COM*адаптер:протокол*" **AS** [#]*номер устройства* [**LEN**=*длина буфера*]

Присваивание заданного *номера файла/устройства адаптеру* последовательной передачи данных с номером 1 или 2. Возможно (хотя и не обязательно) задание протокола передачи данных. Допустимые опции параметра *протокол* перечислены в табл. А.1. (В дисковом и расширенном Бэйсике.)

OPTION BASE *наименьшее значение индекса*

Установка наименьшего значения индекса (0 или 1) для всех массивов (см. DIM). (Во всех разновидностях Бэйсика.)

OUT *порт,байтовое значение*

Запись заданного *байтового значения* в указанный машинный *порт* вывода. (Во всех разновидностях Бэйсика.)

PAINT *(столбец, строка)* [,*цвет-заполнитель*[,*цвет-ограничитель*]]

Закрашивание *цветом-заполнителем* области экрана, начиная с точки с заданными координатами и по всем направлениям. По каждому направлению закрашивание продолжается до тех пор, пока не встретятся точки, окрашенные в заданный *цвет-ограничитель*. В качестве значений необязательных параметров *цвет-заполнитель* и *цвет-ограничитель* указываются 0 или 1 в режиме высокого разрешения или от 0 до 3 в режиме среднего разрешения для выбора одного из четырех цветов активной палитры (см. табл. 13.3). Если указанные параметры не заданы, то при выполнении оператора используется стандартный цвет переднего плана. (В расширенном Бэйсике.)

PEN *действие*

Значение параметра *действие* (OFF или ON) определяет, можно ли использовать функцию PEN (если ON — можно, а если OFF — нельзя). В расширенном Бэйсике в качестве значения этого параметра можно задавать также и STOP; в этом случае ON разблокирует, OFF блокирует, а STOP приостанавливает слежение за работой светового пера. (Во всех разновидностях Бэйсика.)

PLAY *строка подкоманд*

Исполнение встроенным звуковым устройством мелодии, определяемой заданной *строкой подкоманд*. Возможные подкоманды перечислены в табл. 14.2. (В расширенном Бэйсике.)

POKE *смещение,байтовое значение*

Запись заданного *байтового значения* по адресу, определяемому смещением (см. DEF SEG). (Во всех разновидностях Бэйсика.)

PRESET *(столбец, строка)*[,*цвет*]

Построение на экране дисплея отдельной точки. Если *цвет* для нее не задан (выбором по табл. 13.3 одного из цветов активной палитры: 0 или 1 в режиме высокого разрешения, от 0 до 3 в режиме среднего разрешения), то используется цвет фона. (Во всех разновидностях Бэйсика.)

PRINT [*список значений*][;]

Вывод на экран перечисленных в списке значений, если они заданы. Когда в конце оператора не стоит точка с запятой, после вывода последнего значения формируется сигнал возврата каретки и перехода на следующую строку. Точка с запятой, разделяющая значения в списке, не оказывает никакого влияния на то, в какие позиции экранной строки будут помещаться выводимые значения; если же между отдельными значениями в списке стоят запятые, то каждое значение будет выводиться на экран, начиная с первой позиции очередной зоны. (Во всех разновидностях Бэйсика.)

PRINT [#*номер файла/устройства*,] [**USING** *шаблон*;] *список значений*

Вывод перечисленных в списке значений на экран либо, если задано значение первого факультативного параметра, на требуемое устройство или в соответствующий файл. Задавая необязательный параметр *шаблон*, можно выводить данные в нужном формате (допустимые символы, используемые в шаблоне, и их интерпретация приводятся в табл. 10.1). Точка с запятой, разделяющая значения в списке, не оказывает никакого влияния на то, в какие позиции экранной строки будут помещаться выводимые значения. Однако если в операторе отсутствует конструкция USING, то между отдельными значениями в списке можно ставить запятые, и в этом случае каждое значение будет располагаться с начала очередной зоны вывода. Если в конце оператора стоит точка с запятой, то обычного возврата каретки после вывода на экран последнего значения не производится. (Во всех разновидностях Бэйсика.)

PSET *(столбец,строка)*[,*цвет*]

Построение на экране дисплея отдельной точки. Если цвет для нее не задан путем выбора одного из цветов активной палитры по табл. 13.3 (0 или 1 в режиме высокого разрешения, от 0 до 3 в режиме

среднего разрешения), то используется цвет переднего плана. (Во всех разновидностях Бэйсика.)

PUT #*номер файла*[,*номер записи*]

Занесение записи в файл с произвольным доступом. Если номер записи не задан, то ей присваивается очередной номер. (В дисковом и расширенном Бэйсике.)

PUT (*столбец*,*строка*),*имя массива*[,*параметр смеси цветов*]

Воспроизведение на экране цветов всех точек некоторой прямоугольной области, начиная с точки с заданными координатами, которая соответствует верхнему левому углу указанного прямоугольника. Информация о цветах точек хранится в виде кодовых номеров цвета в массиве с заданным именем. Необязательный последний параметр смеси определяет, по каким правилам должно происходить слияние заданных цветов с уже имеющимися на экране. Возможные значения *параметра смеси цветов* перечислены в табл. 13.5. (В расширенном Бэйсике.)

RANDOMIZE *целое значение*

Выбор набора случайных чисел, определяемого заданным целым значением (см. функцию RND). (Во всех разновидностях Бэйсика.)

READ *переменная*[,*переменная*]...

Присваивание переменным с заданными именами значений из списка, созданного операторами DATA. (Во всех разновидностях Бэйсика.)

REM [*комментарий*]

Задание комментария; все символы, следующие за командным словом и стоящие в той же самой программной строке, интерпретируются как комментарий к программе и не исполняются. (Во всех разновидностях Бэйсика.)

RESTORE [*номер строки*]

Восстановление положения указателя списка значений операторов DATA так, что указатель будет соответствовать первому оператору DATA данной программы либо оператору DATA, стоящему в строке с заданным номером (необязательный параметр *номер строки*). (Во всех разновидностях Бэйсика.)

RESUME [*опция*]

Возобновление выполнения программы после выявления и обработки ошибки оператором ON ERROR GOTO. Если необязательный параметр *опция* отсутствует или его значение равно 0, то выполнение возобновляется с того оператора, в котором произошла ошибка. Когда в качестве значения этого параметра указан номер строки, выполнение начинается со строки с этим номером. Если этот параметр принимает значение "NEXT", то выполнение программы про-

должается с оператора, непосредственно следующего за тем, в котором была обнаружена ошибка. (Во всех разновидностях Бэйсика.)

RETURN [*номер строки*]

Возврат управления из подпрограммы оператору, стоящему непосредственно за самым последним выполненным оператором GOSUB (или ON=GOSUB). В расширенном Бэйсике допустим также возврат управления строке с заданным номером (необязательный параметр *номер строки*). (Во всех разновидностях Бэйсика.)

RSET *переменная поля=строковое значение*

Заполнение указанным *строковым значением* заданного поля в файле с произвольным доступом. При заполнении строковое значение выравнивается по крайней правой позиции, а оставшиеся неиспользованными левые позиции поля заполняются пробелами. Производится также выравнивание по правым позициям значения для стандартной строковой переменной. (В дисковом и расширенном Бэйсике.)

SCREEN [*режим*][,[*свечение*][,[*активная страница*][,*видимая страница*]]]

Выбор *режима* работы экрана (значение параметра *режим*, равное 0, определяет режим текстового вывода, 1—режим графического вывода со средней разрешающей способностью, а 2—режим графического вывода с высокой разрешающей способностью). Использование факультативного параметра *свечение* позволяет запретить воспроизведение цветного изображения (при значении этого параметра, равном 0 для режима текстового вывода или <> 0 для режима среднего разрешения). В режиме текстового вывода можно также с помощью соответствующих необязательных параметров установить активную страницу для операторов вывода, подобных PRINT, а также видимую страницу, которая будет выводиться на экран (допустимыми номерами страниц являются номера от 0 до 3 при 80-символьной ширине экрана и от 0 до 8 при 40-символьной ширине). (Во всех разновидностях Бэйсика.)

SOUND *частота,длительность*

Генерация тона заданной частоты (от 37 до 32 767 Гц) и заданной длительности (измеряемой с помощью тактовых импульсов частотой 18,2 имп./с). Значение параметра *длительность*, равное 0, прекращает звучание. В расширенном Бэйсике на выполнение оператора SOUND оказывают влияние некоторые подкоманды, используемые в операторе PLAY. (Во всех разновидностях Бэйсика.)

STOP

Прекращение выполнения программы, вывод на экран сообщения "Break..." («Прерывание...») и возврат к режиму немедленной обработки. (Во всех разновидностях Бэйсика.)

STRIG *действие*

Значение параметра действие (OFF или ON) определяет, может ли использоваться функция STRIG (если ON, то может, а если OFF — нет). (Во всех разновидностях Бэйсика.)

STRIG *(числовое значение) действие*

В зависимости от значения параметра *действие* (OFF, ON или STOP) блокируется (OFF), разблокируется (ON) или приостанавливается (STOP) слежение за заданным триггером управления электронной игрой; триггеры задаются *числовыми значениями* следующим образом: 0 определяет триггер A1, 2 — триггер B1, 4 — триггер A2, 6 — триггер B2. (В расширенном Бэйсике.)

SWAP *переменная,переменная*

Обмен значениями между двумя переменными с заданными именами. (Во всех разновидностях Бэйсика.)

TIME$=*строковое значение*

Установка системного счетчика времени. (В дисковом и расширенном Бэйсике.)

WAIT$ *порт,маска*[,*выборка*]

Приостановка выполнения программы и текущий контроль входного порта. Выполнение программы возобновляется, если следующее выражение имеет ненулевое значение:

порт XOR *выборка* AND *маска*

Если параметр *выборка* отсутствует, то его значение принимается равным 0. (Во всех разновидностях Бэйсика.)

WEND

Возврат управления предыдущему оператору WHILE. (Во всех разновидностях Бэйсика.)

WHILE *условие*

Если заданное *условие* истинно, то выполнение программы продолжается со следующего (за WHILE) оператора. В противном случае управление передается оператору, непосредственно следующему за ближайшим очередным оператором WEND. (Во всех разновидностях Бэйсика.)

WIDTH [*номер устройства*,]*ширина*

Установка длины строки экрана дисплея или какого-либо другого устройства, если задан его номер (параметр *номер устройства*). (Во всех разновидностях Бэйсика.)

WIDTH *имя устройства,ширина*

Подготовка к изменению длины строки для устройства с заданным именем; указанная длина устанавливается не сразу, а только после открытия соответствующего устройства. (Во всех разновидностях Бэйсика.)

WRITE (#*номер файла*,] *список значений*

Вывод на экран перечисленных в списке значений; между отдельными значениями выводятся запятые, а строковые значения заключаются в кавычки. Если задан необязательный параметр *номер файла*, то значения записываются аналогичным образом в файл с указанным номером. (Во всех разновидностях Бэйсика.)

Функции

Все функции перечисляются в алфавитном порядке. Большинство функций имеет один или два операнда, в качестве которых могут использоваться константы, переменные, элементы массивов, другие функции и выражения, если специально не оговорена иная интерпретация. Для некоторых функций ограничивается диапазон изменения значений их операндов. Результатами вычисления функций являются числа с обычной точностью, если на этот счет нет каких-либо иных указаний.

ABS(*числовое значение*)

Вычисление абсолютной величины *числового значения*. (Во всех разновидностях Бэйсика.)

ASC(*строковое значение*)

Определение кода первого символа заданного *строкового значения*. (Во всех разновидностях Бэйсика.)

ATN(*числовое значение*)

Вычисление арктангенса заданного *числового значения*. (Во всех разновидностях Бэйсика.)

CDBL(*числовое значение*)

Преобразование заданного *числового значения* в число удвоенной точности. (Во всех разновидностях Бэйсика.)

CHR$(*код*)

Определение символа, соответствующего заданному *коду* (см. Приложение D). (Во всех разновидностях Бэйсика.)

CINT (*числовое значение*)

Округление заданного *числового значения* до ближайшего целого числа. (Во всех разновидностях Бэйсика.)

COS(*числовое значение*)

Вычисление косинуса заданного *числового значения*. (Во всех разновидностях Бэйсика.)

CSNG(*числовое значение*)

Преобразование заданного *числового значения* в число обычной точности. (Во всех разновидностях Бэйсика.)

CSRLIN

Выдача номера экранной строки, соответствующей текущему положению курсора (в режиме текстового вывода). (Во всех разновидностях Бэйсика.)

CVD*(строковое значение)*

Преобразование заданного восьмисимвольного *строкового значения* в значение с двойной точностью (функция, обратная по отношению к MKD$). (В дисковом и расширенном Бэйсике.)

CVI*(строковое значение)*

Преобразование заданного двухсимвольного *строкового значения* в численное значение целого типа (функция, обратная по отношению к MKI$). (В дисковом и расширенном Бэйсике.)

CVS*(строковое значение)*

Преобразование заданного четырехсимвольного *строкового значения* в значение с обычной точностью (функция, обратная по отношению к MKS$). (В дисковом и расширенном Бэйсике.)

DATE$

Выдача текущей системной даты. (В дисковом и расширенном Бэйсике.)

EOF*(номер файла)*

Выдача значения «Истина» (—1) или «Ложь» (0) в зависимости от того, был ли достигнут конец заданного файла или нет. (Во всех разновидностях Бэйсика.)

ERL

Выдача номера строки, в которой обнаружена последняя по счету ошибка. (Во всех рановидностях Бэйсика.)

ERR

Выдача кода последней по счету обнаруженной ошибки. (См. Приложение С). (Во всех разновидностях Бэйсика.)

EXP*(числовое значение)*

Возведение константы *е* (2,718282) в степень, равную заданному *числовому значению*. (Во всех разновидностях Бэйсика.)

FIX*(числовое значение)*

Преобразование заданного *числового значения* в целое число путем отбрасывания дробной части. (Во всех разновидностях Бэйсика.)

FN*имя* [(*значение* [,*значение*]...)]

Вызов названной функции, предварительно определенной оператором DEF FN, который одновременно задает количество и типы значений, необходимых для вызова этой функции. (Во всех разновидностях Бэйсика.)

FRE*(строковое значение или числовое значение)*

Выдача сообщения об объеме свободной памяти. Задаваемое значение как таковое нигде не используется, но его присутствие вызывает реорганизацию области памяти, предназначенной для хранения строковых данных. (Во всех разновидностях Бэйсика.)

HEX$*(числовое значение)*

Преобразование заданного *числового значения* в его шестнадцатеричный эквивалент. (Во всех разновидностях Бэйсика.)

INKEY$

Выдача информации о том, какая клавиша нажимается в текущий момент. Нулевое значение в качестве результата функции означает, что клавиатура блокирована. Односимвольное значение результата интерпретируется как символ, соответствующий нажимаемой клавише. Если же нажимаемая клавиша не имеет соответствующего ей символа, то выдается двухсимвольное значение, которое должно рассматриваться как расширенный код для этой клавиши (см. Приложение D). (Во всех разновидностях Бэйсика.)

INP*(порт)*

Выдача сообщения о назначенном машинном *порте*. (Во всех разновидностях Бэйсика.)

INPUT$*(число*[,[#] *номер файла/устройства*])

Считывание заданного числа символов с клавиатуры либо, если задан параметр *номер файла/устройства*, с соответствующего устройства или из файла. (Во всех разновидностях Бэйсика.)

INSTR([*начальный символ*],*исходная строка*,*искомая строка*)

Поиск в исходной строке первого вхождения искомой строки, начиная с заданного начального символа, если он указан, или с самого первого символа исходной строки. (Во всех разновидностях Бэйсика.)

INT*(числовое значение)*

Определение наибольшего целого числа, не превосходящего заданное *числовое значение*. (Во всех разновидностях Бэйсика.)

LEFT$*(строковое значение*,*длина)*

Выделение из *строкового значения* подстроки заданной *длины*, начиная с крайнего левого символа. (Во всех разновидностях Бэйсика.)

LEN*(строковое значение)*

Подсчет количества символов в заданном *строковом значении*. (Во всех разновидностях Бэйсика.)

LOC*(номер файла)*

Определение текущей позиции в заданном файле. (В дисковом и расширенном Бэйсике.)

LOF(*номер файла*)

Выдача сведений о длине заданного файла. (В дисковом и расширенном Бэйсике.)

LOG(*числовое значение*)

Вычисление натурального логарифма заданного числа. (Во всех разновидностях Бэйсика.)

LPOS(*числовое значение*)

Выдача информации о знакопозиции последнего символа в буфере печатающего устройства; в кассетном Бэйсике соответствующее число должно быть задано, хотя оно и не используется. В дисковом и расширенном Бэйсике заданное *числовое значение* определяет используемое печатающее устройство (0 соответствует устройству LPT1:, 1 — LPT1:, 2 — LPT2; 3 — LPT3:). (Во всех разновидностях Бэйсика.)

MID$(*строковое значение,начальный символ*,[*длина*])

Выделение части *строкового значения*, начинающейся с первого вхождения данного начального символа. Задавая необязательный параметр *длина*, можно ограничить число символов выделяемой подстроки. (Во всех разновидностях Бэйсика.)

MKD$(*числовое значение*)

Преобразование заданного *числового значения* в число удвоенной точности, а затем представление его в виде восьмисимвольного строкового значения (функция, обратная по отношению к CVD). (В дисковом и расширенном Бэйсике.)

MKI$(*числовое значение*)

Округление заданного *числа* до целого и представление последнего в виде двухсимвольного строкового значения (функция, обратная по отношению к CVI). (В дисковом и расширенном Бэйсике.)

MKS$(*числовое значение*)

Преобразование заданного *числового значения* в число обычной точности и представление последнего в виде четырехсимвольного строкового значения (функция, обратная по отношению к CVS). (В дисковом и расширенном Бэйсике.)

OCT$(*числовое значение*)

Представление заданного *числового значения* в восьмеричной форме. (Во всех разновидностях Бэйсика.)

PEEK(*смещение*)

Выдача содержимого ячейки памяти по заданному *смещению* и текущему адресу сегмента (см. оператор DEF SEG). (Во всех разновидностях Бэйсика.)

PEN(*числовое значение*)

Выдача информации о работе светового пера. Интерпретация раз-

личных задаваемых *чисел* приводится в табл. А.2. Функцию PEN можно использовать только после того, как выполнен оператор PEN ON. (Во всех разновидностях Бэйсика.)

POINT(*столбец,строка*)
Определение цвета точки с заданными координатами (должна использоваться система координат, принятая для графического режима). (Во всех разновидностях Бэйсика.)

POS(*числовое значение*)
Выдача номера столбца, соответствующего текущему положению курсора на экране в текстовом режиме вывода. Задаваемое число при выполнении функции не используется, но обязательно должно быть указано. (Во всех разновидностях Бэйсика.)

RIGHT$(*строковое значение,длина*)
Выделение подстроки заданной *длины*, начиная с крайнего правого символа исходного *строкового значения*. (Во всех разновидностях Бэйсика.)

RND[(*числовое значение*)]
Выдача случайного числа, заключенного между 0 и 1. Если заданное в функции *числовое значение* равно 0, то повторно выдается то же случайное число, что и в предыдущем обращении к этой функции. Если *числовое значение* положительно или отсутствует, то выбирается очередное число из текущего списка случайных чисел. Если *числовое значение* отрицательно, то оно определяет новый список случайных чисел, из которого выбирается первый элемент (см. оператор RANDOMIZE). (Во всех разновидностях Бэйсика.)

SCREEN(*строка,столбец*[,*опция*])
Определение числового кода символа, высвеченного на экране в заданной позиции (при задании позиции должна использоваться нумерация строк и столбцов экрана, принятая в текстовом режиме). Параметр *опция* имеет смысл только в текстовом режиме. Если этот параметр принимает ненулевое значение («Истина»), то вместо числового кода символа, находящегося в заданной позиции, функция SCREEN выдает атрибуты цвета для этой позиции на экране. Ниже приводятся выражения, с помощью которых расшифровывается значение атрибутов цвета. (Во всех разновидностях Бэйсика.)

атрибуты цвета=SCREEN(*строка,столбец*, —1)

цвет переднего плана-атрибуты цвета MOD 16

цвет фона=((*атрибуты цвета* — *цвет переднего плана*)/16) MOD 128

мерцающие символы, если условие (*атрибуты цвета*>127) истинно (—1)

SGN*(числовое значение)*

Выдача знака заданного *числового значения*: результат выполнения функции равен +1, если *числовое значение* положительно; —1, если оно отрицательно, и 0, если оно равно 0. (Во всех разновидностях Бэйсика.)

SIN*(числовое значение)*

Вычисление синуса заданной числовой величины. (Во всех разновидностях Бэйсика.)

SPACE$*(число)*

Генерация заданного *числа* пробелов. (Во всех разновидностях Бэйсика.)

SPC*(число)*

Пропуск заданного *числа* позиций в выводимой на печать строке. Разрешается использовать только вместе с операторами PRINT и LPRINT. (Во всех разновидностях Бэйсика.)

SQR*(числовое значение)*

Вычисление квадратного корня из заданного числа. (Во всех разновидностях Бэйсика.)

STICK*(числовое значение)*

Выдача информации о положении двух ручек или иных органов управления электронными играми. Действительная проверка позиций органов управления производится только тогда, когда задаваемое в качестве аргумента *числовое значение* равно 0, однако и при других значениях выдаются координаты по отдельным направлениям. Допустимые числовые значения представлены в табл. А.3. (Во всех разновидностях Бэйсика.)

STRIG*(числовое значение)*

Определение состояния триггеров (кнопок) управления играми. В зависимости от конкретного *числового значения* операнда проверяется тот или иной триггер. Соответствие между возможными численными значениями и триггерами определяется табл. А.4. Функция STRIG может выполняться только после того, как был выполнен оператор STRIG ON. (Во всех разновидностях Бэйсика.)

STRING$*(длина, строковое значение или код)*

Генерация строки заданной *длины*. Все символы этой строки одинаковы и совпадают с первым символом заданного строкового значения либо с символом, имеющим заданный числовой код, в зависимости от того, какой из этих параметров указан. (Во всех разновидностях Бэйсика.)

STR$*(числовое значение)*

Преобразование заданного *числового значения* в строку символов. (Во всех разновидностях Бэйсика.)

TAB*(столбец)*

Подведение к заданной позиции (столбцу) в выводимой на печать строке. Функция имеет смысл только вместе с операторами PRINT и LPRINT. (Во всех разновидностях Бэйсика.)

TAN*(числовое значение)*

Вычисление тангенса заданной числовой величины. (Во всех разновидностях Бэйсика.)

TIME$

Выдача системного времени. (В дисковом и расширенном Бэйсике.)

USR[*номер*]*(строковая переменная* или *числовое значение)*

Передача управления программе на машинном языке, находящейся по адресу, задаваемому оператором DEF USR. Программе на машинном языке передается либо числовое значение, либо адрес памяти, где хранится строковая переменная, в зависимости от того, какой из двух параметров задан *(строковая переменная* или *числовое значение)*. (Во всех разновидностях Бэйсика.)

VAL*(строковое значение)*

Преобразование *строкового значения* в число. (Во всех разновидностях Бэйсика.)

VARPTR*(переменная)*

Определение адреса памяти, по которому хранится значение заданной переменной. (Во всех разновидностях Бэйсика.)

VARPTR(#*номер файла*)

Определение адреса памяти, по которому хранится блок управления файлами Бэйсик-интерпретатора для заданного файла. (Во всех разновидностях Бэйсика.)

VARPTR$*(переменная)*

Определение типа переменной и соответствующего ей адреса памяти. Эти данные выдаются в виде трехсимвольного строкового значения. (В дисковом и расширенном Бэйсике.)

Производные функции

Путем комбинации стандартных трансцендентных функций можно получать различные другие функции того же типа, перечисленные ниже. Вывод этих функций определяется уравнениями относительно *n*, которое может представлять собой числовую константу, переменную, элемент массива, функцию или выражение. Дополнительно отмечены все значения аргумента *n*, использование которых приводит к сообщениям об ошибках.

ARCCOS(n)=1.570796 — ATN (n/SQR (1—n*n)) — функция, обратная косинусу n, где ABS(n)<1.

ARCCOT(n)=1.570796 — ATN(n) — функция, обратная котангенсу n.

ARCCOSH(n)=LOG(n+SQR(n*n—1)) — функция, обратная гиперболическому косинусу n, где n > =1.

ARCCOTH(n)=LOG((n+1)/(n—1))/2 — функция, обратная гиперболическому котангенсу n, где ABS(n)>1.

ARCCSC(n)=ATN(1/SQR(n*n—1))+(SGN(n)—1)*1.570796 — функция, обратная косекансу n, где ABS(n)>1.

ARCCSCH(n)=LOG((SGN(n)*SQR(n*n+1)+1)/n) — функция, обратная гиперболическому косекансу n, где n>0.

ARCSEC(n)=ATN((SQR(n*n—1))+(SGN(n) — 1)*1.570796 — функция, обратная секансу n, где ABS(n)>=1.

ARCSECH(n)=LOG((SQR(1—n*n)+1)/n) — функция, обратная гиперболическому секансу n, где 0<n<=1.

ARCSIN(n)=ATN(n/SQR(1—n*n)) — функция, обратная синусу n, где ABS(n)<1.

ARCSINH(n)=LOG(n+SQR(n*n+1)) — функция, обратная гиперболическому синусу n.

ARCTANH(n)=LOG((1+n)/(1—n))/2 — функция, обратная гиперболическому тангенсу n, где ABS(n)<1.

COSH(n)=(EXP(n)+EXP(—n))/2 — гиперболический косинус n.

COT(n)=1/TAN(n) — котангенс n, где n<>0.

COTH(n)=EXP(—n)/EXP(n)—EXP(—n))*2+1 — гиперболический котангенс n, где n<>0.

CSC(n)=1/SIN(n) — косеканс n, где n<>0.

CSCH(n)=2/(EXP(n) — EXP(—n)) — гиперболический косеканс n, где n<>0.

LOG_a (n)=LOG(n)/LOG(a) — логарифм n по основанию a, где a>0 и n>0.

LOG_{10}(n)=LOG(n)/2.302585 — десятичный логарифм n, где n>0.

SEC(n)=1/COS(n) — секанс n, где n<>1.570796.

SECH(n)=2/(EXP(n)+EXP(—n)) — гиперболический секанс n.

SINH(*n*)=(EXP(*n*)—EXP(—*n*))/2 — гиперболический синус *n*.

TANH(*n*)=(EXP(—*n*)—EXP(*n*))/EXP(—*n*))+EXP(*n*)) — гиперболический тангенс *n*.

Таблица А.1. **Опции протокола последовательной передачи данных** [1]

Опция	Интерпретация
бод	Скорость передачи данных: 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600
способ контроля	S—по пробелам (0 бит), M—по маркеру (1 бит), O—по нечетности, E—по четности, N—в случае отсутствия контроля
,*слово*	Длина информационного слова в битах: 4, 5, 6, 7 или 8
,*стоп*	Число стоповых битов: 1 или 2
,RS	Подавление сигнала запроса на передачу (RTS)
,CS*мс*	Ожидание в течение заданного числа миллисекунд сигнала сброса (CTS)
,DS*мс*	Ожидание в течение заданного числа миллисекунд сигнала готовности набора данных (DSR)
,CD*мс*	Ожидание в течении заданного числа миллисекунд сигнала несущей (CD)
,LF	Принудительное формирование символа перевода строки после каждого символа возврата каретки

[1]) Данные опции используются в операторах OPEN "COM и задаются в том порядке, как они перечислены в таблице.

Таблица А.2. **Операнд функции PEN**

Численное значение операнда	Результат выполнения
0	Световое перо выключено с момента последнего опроса [1]
1	Номер столбца, который последний раз отмечался световым пером в графическом режиме
2	Номер строки, которая последний раз отмечалась световым пером в графическом режиме
3	Световое перо включено [1]
4	Номер последнего допустимого столбца в графическом режиме
5	Номер последней допустимой строки в графическом режиме
6	Номер строки, которая последний раз отмечалась световым пером в текстовом режиме
7	Номер столбца, который последний раз отмечался световым пером в текстовом режиме
8	Номер последней допустимой строки в текстовом режиме
9	Номер последнего допустимого столбца в текстовом режиме

[1]) Результатами выполнения функций PEN(0) и PEN(3) являются значения «Истина» (—1) и «Ложь» (0).

Таблица А.3. **Операнд функции STICK**

Численное значение операнда	Результат выполнения
0	Горизонтальная координата, ручка управления А
1	Вертикальная координата, ручка управления А
2	Горизонтальная координата, ручка управления В
3	Вертикальная координата, ручка управления В

Таблица А.4. **Операнд функции STRIG**

Численное значение операнда [1]	Результаты выполнения [2]
0	Клавиша триггера А1 нажата с момента последнего выполнения функции STRIG(0)
1	Клавиша триггера А1 нажимается в данный момент
2	Клавиша триггера В1 нажата с момента последнего выполнения функции STRIG(2)
3	Клавиша триггера В1 нажимается в данный момент
4	Клавиша триггера А2 нажата с момента последнего выполнения функции STRIG(4)
5	Клавиша триггера А2 нажимается в данный момент
6	Клавиша триггера В2 нажата с момента последнего выполнения функции STRIG(6)
7	Клавиша триггера В2 нажимается в данный момент

[1]) Численные значения 4—7 имеют смысл только в расширенном Бэйсике.
[2]) Результатом выполнения функции STRIG всегда является значение «Истина» (—1) или «Ложь» (0).

Приложение В

КРАТКОЕ ОПИСАНИЕ КОМАНД ДОС ПВМ

В данном приложении дается описание всех стандартных команд ДОС ПВМ (версия 1.10), перечисленных в алфавитном порядке. Более подробную информацию можно найти в гл. 3 и 5 данной книги или в гл. 3 книги IBM "Personal Computer Disk Operating System", 2nd ed., содержащей описание дисковой операционной системы для ПВМ.

Поскольку в ПВМ предусматриваются средства формирования пользовательских команд (гл. 5), каждая конкретная вычислительная система может иметь команды, не перечисленные в данном приложении. Так, например, для винчестерских накопителей часто используются специализированные команды форматирования и копирования дисков.

При вводе рассматриваемых ниже команд с клавиатуры следует набирать все символы, выделенные ЖИРНЫМ ШРИФТОМ, в точности так, как они напечатаны; разрешается лишь в любом месте заменять прописные буквы на строчные и наоборот. Все, что напечатано курсивом, должно заменяться на фактические значения. Текст, напечатанный в квадратных скобках, является необязательным, однако если он используется в команде, то сами квадратные скобки набирать не следует. Любой параметр, за которым следует многоточие (. . .), можно повторять многократно, причем само многоточие при этом не набирается. Необходимо внимательно следить за тем, чтобы при наборе команд вводились все указанные здесь знаки препинания (кроме квадратных скобок и многоточия), включая запятые, двоеточия, косую черту, знаки равенства и знаки плюс.

дисковод:

Регистрация заданного *дисковода* как используемого по умолчанию в тех командах, где опущено явное определение дисковода.

[*дисковод*:]*файл*[*опции*]

Работа с командным или пакетным *файлом*. Количество и вид задаваемых *опций* определяются конкретным командным или пакетным *файлом*.

CHKDSK [*дисковод*:]

Проверка целостности дисковых файлов и выдача информации об использовании дисковой памяти.

COMP [*дисковод*:] *файл* [*дисковод*:][*файл*]

Сравнение содержимого двух файлов с заданными именами и выдача информации о местонахождении обнаруженных несовпадений.

COPY [*дисковод*:]*исходный файл* [*опция*] [+[*дисковод*:] *присоединяемый файл* [*опция*]][*дисковод*:][*результирующий файл*][*опция*][/V]

Копирование заданного *исходного* файла в *результирующий* с возможностью соединения исходного файла с другим *присоединяемым файлом*. Если указан факультативный параметр /V, то производится проверка полученной копии. В качестве значения параметра *опция* может быть задано /A или /B. В случае опции /A исходный файл копируется до первого символа конца файла (код 26), и этот символ добавляется в конец результирующего файла. Если имеет место опция /B, то исходный файл копируется целиком, а в конец результирующего файла не добавляется никаких специальных символов.

DATE [*дата*]

Установка системной даты. Если параметр *дата* не задан, формируется запрос на его ввод.

DIR [*дисковод*:][*файл*] [/P][/W]

Вывод всего дискового справочника или некоторой его части.

DISKCOMP [*дисковод*:] [*дисковод*:] [/1]

Сравнение содержимого двух дискетов и выдача информации обо всех обнаруженных несоответствиях. Если задан необязательный параметр /1, то дискеты сравниваются только по одной стороне.

DISKCOPY [*исходный дисковод*:][*результирующий дисковод*:] [/1]

Копирование дискета, установленного на заданном *исходном* дисководе, на дискет *результирующего* дисковода. Когда задан только один дисковод (или ни одного), копирование производится на тот же носитель. Если задан параметр /1, то копируется лишь одна сторона дискета.

ERASE [*дисковод*:]*файл*

Удаление из дискового справочника файла с заданным именем.

EXE2BIN [*дисковод*:]*файл* [*дисковод*:][*файл*]

Преобразование файла типа EXE в файл типа COM.

FORMAT [*дисковод*:] [/S][/1]

Инициализация дискета с форматом хранения записей, принятым в ДОС ПВМ, выделение и отметка на дискете дефектных областей, разметка незаполненного справочника. При задании опции /S форматированный дискет назначается системным диском, а в случае опции /1 форматируется лишь одна сторона дискета.

MODE LPT:*номер*[*длина*][,*высота*]

Установка *длины* строки (от 1 до 132 символов) и *высоты* шрифта (1/6 или 1/8 дюйма) печатающего устройства под номером 1, 2 или 3. Задание высоты строки возможно лишь в некоторых моделях печатающих устройств.

MODE[*ширина*][,*направление*][,T]

Установка *ширины* рабочего поля экрана (40 или 80 символов). Можно также по этой команде сдвинуть изображение на экране в заданном направлении (значение параметра *направление*, равное R, соответствует сдвигу вправо, а L — влево). Если опция T задана, то изображение сдвигается вместе с текстовыми данными; если T отсутствует, то — без них.

MODE COM*адаптер*:*скорость*[,*способ контроля*[,*слово*[,*стоповая посылка* [,**P**]]]]

Определение протокола последовательной передачи данных, который должен будет использоваться с *адаптером* 1 или 2. Скорость передачи данных может принимать значения 110, 150, 300, 600, 1200, 2400, 4800 или 9600 бод. Контроль, определяемый значением параметра *способ контроля*, может быть по четности (E), по нечетности (O) или может вообще отсутствовать (N). Длина слова может составлять 7 или 8 бит. Могут быть заданы 1 или 2 стоповых бита.

MODE LPT*номер*:=**COM***адаптер*

Переадресация выхода для печатающего устройства под *номером* 1, 2 или 3 (значение параметра *номер*) на *адаптер* последовательной передачи данных номер 1 или 2.

PAUSE [*комментарий*]

Приостановка обработки для вывода на экран слова "PAUSE" вместе с *комментарием*, если он задан, а также сообщения "Strike any key when ready..." («По готовности нажмите любую клавишу...»), после чего система ждет нажатия какой-либо клавиши.

REM [*комментарий*]

Вывод на экран слова "REM" вместе с комментарием, если он задан.

RENAME [*дисковод*:]*старое имя новое имя*

Замена *старого* имени файла *новым*.

SYS *дисковод*:

Копирование программных файлов ДОС ПВМ, необходимых для превращения диска, установленного на заданном дисководе, в системный; при этом устанавливаемый диск должен быть размечен с использованием опции /S в команде FORMAT.

TIME [*время*]

Установка системного времени. Если параметр *время* отсутствует, то ДОС ПВМ формирует запрос на ввод его значения.

TYPE [*дисковод*:]*файл*

Вывод на экран содержимого заданного *файла*; при выводе содержимое интерпретируется как набор символов в коде ASCII (см. Приложение D).

Приложение С

СООБЩЕНИЯ ОБ ОШИБКАХ

В данном приложении содержится краткое описание всех сообщений об ошибках, выдаваемых интерпретатором Бэйсика ПВМ. Сообщения перечисляются в алфавитном порядке; для каждого из них указан код ошибки, выдаваемый функцией ERL. Этот кодовый номер служит также перекрестной ссылкой для списка развернутых сообщений об ошибках, содержащегося в приложении А книги "IBM Personal Computer BASIC", 2nd ed.

Сообщение	Перевод	Кодовый номер ошибки
Advanced feature	Средство расширенного Бэйсика	73
Bad file mode	Неправильный режим файла	54
Bad file name	Неверное имя файла	64
Bad file number	Неверный номер файла	52
Bad record number	Неверный номер записи	63
Can't continue	Продолжение невозможно	17
Communication buffer overflow	Переполнение буфера информационного обмена	69
Device Fault	Сбой в работе устройства	25
Device I/O error	Ошибка ввода-вывода	57
Device Timeout	Устройство выключено из работы	24
Device Unavailable	Устройство недоступно	68
Direct statement in file	Непосредственный оператор в файле	66
Disk full	Диск заполнен	61
Disk Media Error	Дефект диска	72
Disk not Ready	Диск не готов	71
Disk Write Protect	Запрещена запись на диск	70
Division by zero	Деление на ноль	11
Duplicate definition	Повторное определение	10
FIELD overflow	Переполнение при выполнении оператора FIELD	50
File already exists	Файл уже существует	58
File already open	Файл уже открыт	55
File not found	Файл уже найден	53
FOR without NEXT	FOR без NEXT	26
Illegal direct	Недопустимая директива	12
Illegal function call	Недопустимый вызов функции	5
Input past end	Ввод после признака конца	62
Internal error	Внутренняя ошибка	51
Line buffer overflow	Переполнение буфера строки	23
Missing operand	Отсутствует операнд	22
NEXT without FOR	NEXT без FOR	1

Продолжение

Сообщение	Перевод	Кодовый номер ошибки
No RESUME	Нет оператора RESUME	19
Out of data	Не хватает данных	4
Out of memory	Не хватает памяти	7
Out of paper	Не хватает бумаги	27
Out of string space	Не хватает места в строковой переменной	14
Overflow	Переполнение	6
RESUME without error	RESUME без ON-ERROR	20
RETURN without GOSUB	RETURN без GOSUB	3
String formula too complex	Сложность строковой формулы превышает допустимую	16
String too long	Длина символьной строки превышает допустимую	15
Subscript out of range	Индекс лежит вне пределов установленного диапазона значений	9
Syntax error	Синтаксическая ошибка	2
Too many files	Количество файлов превышает допустимое	67
Type mismatch	Несоответствие типов	13
Undefined line number	Неопределенный номер строки	8
Undefined user function	Неопределенная функция пользователя	18
Unprintable error	Не выводимая на печать ошибка	—
WEND without WHILE	WEND без WHILE	30
WHILE without WEND	WHILE без WEND	29

Приложение D

СИМВОЛЫ, КОДЫ И СПЕЦИАЛЬНЫЕ КЛАВИШИ

В табл. D.1 устанавливается соответствие между 256 символами, которые можно использовать при работе с экраном дисплея, и их числовыми кодами. Коды 128—256 недопустимы в кассетном Бэйсике.

Для того чтобы сгенерировать любой символ с кодом 32—126, достаточно нажать соответствующую клавишу точно так, как это делается на обычной пишущей машинке. Например, нажатие клавиши A вызывает появление на экране буквы A. Символ можно также сгенерировать, если сначала нажать клавишу Alt, а затем, не отпуская ее, набрать на малой цифровой клавиатуре код этого символа (гл. 7). Указанные способы генерации символов не применимы для кодов 0—31 и кода 127 при вводе с клавиатуры команд в режиме немедленной обработки и при ответах на запросы операторов INPUT и LINE INPUT. Однако эти способы можно использовать при вводе ответов на запросы функций INPUT$ и INKEY$. Безусловно, любой допустимый символ можно сгенерировать с помощью функции CHAR$.

Для большинства печатающих устройств коды 0—32 являются управляющими и не имеют внешнего представления. Интерпретация почти всех управляющих кодов меняется в зависимости от конкретной модели печатающего устройства; те немногие из них, которые имеют одну и ту же интерпретацию для всех печатающих устройств, перечислены в табл. 10.3. Интерпретация кодов, больших 127, также в основном зависит от конкретного печатающего устройства, а на печатающих устройствах со съемными шрифтоносителями (лепестковыми или цилиндрическими) даже коды 32—127 могут отличаться по интерпретации от стандарта ASCII, представленного в табл. D.1.

В табл. D.2 перечислены расширенные коды, которые в основном генерируются путем одновременного нажатия нескольких клавиш, включая Alt, ⌂ и Ctrl. Этим кодам не соответствуют никакие символы и их можно выявить только с помощью функции INKEY$ (гл. 11).

Таблица D.1. **Символы экрана и их коды**

Десятичный код	Символ	Десятичный код	Символ	Десятичный код	Символ
000	(нуль)	047	/	094	^
001	☺	048	0	095	←
002	☻	049	1	096	`
003	♥	050	2	097	a
004	♦	051	3	098	b
005	♣	052	4	099	c
006	♠	053	5	100	d
007	(звуковой сигнал)	054	6	101	e
008	(возврат на 1 позиц.)	055	7	102	f
009	(метка табуляции)	056	8	103	g
010	(перевод строки)	057	9	104	h
011	(исх. позиц. курсора)	058	:	105	i
012	(очистка экрана)	059	;	106	j
013	(возврат каретки)	060	<	107	k
014	♫	061	=	108	l
015	☼	062	>	109	m
016	►	063	?	110	n
017	◄	064	@	111	o
018	↕	065	A	112	p
019	‼	066	B	113	q
020	¶	067	C	114	r
021	§	068	D	115	s
022	▬	069	E	116	t
023	↨	070	F	117	u
024	↑	071	G	118	v
025	↓	072	H	119	w
026	→	073	I	120	x
027	←	074	J	121	y
028	(перем. курс. вправо)	075	K	122	z
029	(перем. курс. влево)	076	L	123	{
030	(перем. курс. вверх)	077	M	124	¦
031	(перем. курс. вниз)	078	N	125	}
032	(пробел)	079	O	126	~
033	!	080	P	127	⌂
034	"	081	Q	128	Ç
035	#	082	R	129	ü
036	$	083	S	130	é
037	%	084	T	131	â
038	&	085	U	132	ä
039	'	086	V	133	à
040	(	087	W	134	å
041	)	088	X	135	ç
042	*	089	Y	136	ê
043	+	090	Z	137	ë
044	,	091	[	138	è
045	-	092	\	139	ï
046	.	093	]	140	î

Десятичный код	Символ	Десятичный код	Символ	Десятичный код	Символ
141	ì	180	┤	219	■
142	Ä	181	╡	220	▄
143	Å	182	╢	221	▌
144	É	183	╖	222	▐
145	æ	184	╕	223	▀
146	Æ	185	╣	224	α
147	ê	186	═	225	ß
148	ö	187	╗	226	Γ
149	ò	188	╝	227	π
150	û	189	╜	228	Σ
151	ù	190	╛	229	σ
152	v	191	┐	230	µ
153	Ö	192	└	231	τ
154	Ü	193	┴	232	Φ
155	¢	194	┬	233	ϕ
156	£	195	├	234	Ω
157	¥	196	─	235	δ
158	Pt	197	┼	236	∞
159	ƒ	198	╞	237	∅
160	á	199	╟	238	∈
161	í	200	╚	239	∩
162	ó	201	╔	240	≡
163	ú	202	╩	241	±
164	ñ	203	╦	242	≥
165	Ñ	204	╠	243	≤
166	ª	205	═	244	⌠
167	º	206	╬	245	⌡
168	¿	207	╧	246	÷
169	⌐	208	╨	247	≈
170	¬	209	╤	248	°
171	½	210	╥	249	∙
172	¼	211	╙	250	·
173	¡	212	╘	251	√
174	«	213	╒	252	n
175	»	214	╓	253	²
176	▒	215	╫	254	■
177	▒	216	╪	255	(пробел 'FF')
178	▓	217	┘		
179	│	218	┌		

*) Непосредственный ввод с клавиатуры кода 3 (одновременно с нажатием клавиши Alt) вызывает прерывание аналогично тому, как это происходит при совместном нажатии клавиш Ctrl| Scroll Lock однако функция CHR$ (3) всегда генерирует указанный в таблице символ с кодом 3.

+) Для большинства печатающих устройств код 12 соответствует прогону бумаги к началу следующей страницы.

Таблица D.2. **Коды символов расширенной клавиатуры и соответствующие им комбинации клавиш**

Десятичный код	*Комбинация клавиш*	*Десятичный код*	*Комбинация клавиш*
3	*(нулевой символ)*		
15	⇤	85	⇧|F2
16	Alt|Q	86	⇧|F3
17	Alt|W	87	⇧|F4
18	Alt|E	88	⇧|F5
19	Alt|R	89	⇧|F6
20	Alt|T	90	⇧|F7
21	Alt|Y	91	⇧|F8
22	Alt|U	92	⇧|F9
23	Alt|I	93	⇧|F10
24	Alt|O	94	Ctrl|F1
25	Alt|P	95	Ctrl|F2
30	Alt|A	96	Ctrl|F3
31	Alt|S	97	Ctrl|F4
32	Alt|D	98	Ctrl|F5
33	Alt|F	99	Ctrl|F6
34	Alt|G	100	Ctrl|F7
35	Alt|H	101	Ctrl|F8
36	Alt|J	102	Ctrl|F9
37	Alt|K	103	Ctrl|F10
38	Alt|L	104	Alt|F1
44	Alt|Z	105	Alt|F2
45	Alt|X	106	Alt|F3
46	Alt|C	107	Alt|F4
47	Alt|V	108	Alt|F5
48	Alt|B	109	Alt|F6
49	Alt|N	110	Alt|F7
50	Alt|M	111	Alt|F8
59	F1*	112	Alt|F9
60	F2*	113	Alt|F10
61	F3*	114	Ctrl|PrtSc
62	F4*	115	Ctrl|← *(предыд. слово)*
63	F5*	116	Ctrl|→ *(следующ. слово)*
64	F6*	117	Ctrl|End
65	F7*	118	Ctrl|PgDn
66	F8*	119	Ctrl|Home
67	F9*	120	Alt|1
68	F10*	121	Alt|2
71	Home	122	Alt|3
72	↑	123	Alt|4
73	PgUp	124	Alt|5
75	←	125	Alt|6
77	→	126	Alt|7
79	End	127	Alt|8
80	↓	128	Alt|9
81	PgDn	129	Alt|0
82	Ins	130	Alt|-
83	Del	131	Alt|=
84	⇧|F1	132	Ctrl|PgUp

*) Если не запрограммированы как функциональные клавиши.

СПРАВОЧНАЯ КАРТА ДЛЯ ПВМ ФИРМЫ IBM

Как пользоваться картой

- Текст, напечатанный жирным шрифтом, набирать на клавиатуре дословно.
- Текст, напечатанный курсивом, заменять на фактические значения, переменные или выражения.
- Квадратные скобки ([ ]) означают, что заключенные в них параметры необязательны.
- Многоточие (...) означает, что стоящие перед ним параметры могут повторяться.
- Все остальные знаки препинания набирать на клавиатуре в точности так, как они напечатаны.
- Более подробное описание перечисленных в карте понятий, а также описание не указанных здесь команд, операторов и функций приводится в приложениях А и В.

Краткое описание Бэйсика

- Данное краткое описание Бэйсика применимо для кассетного Бэйсика версии 1.0 и для дискового и расширенного Бэйсика версии 1.10.

Команды Бэйсика

AUTO [*номер первой строки*][,[*приращение*]]
Установка режима автоматической нумерации программных строк.
DELETE *номер первой строки*[— *номер последней строки*]
Удаление одной или более программных строк.
EDIT *номер строки*
Вывод на экран программной строки для редактирования.
FILES [*имя файла*]
Вывод на экран всего дискового справочника или его части.
LIST [*номер строки*][— [*номер строки*]][,*имя файла/устройства*]
Вывод на экран программных строк с номерами из заданного диапазона или вывод их на печать на устройство с заданным именем.
LLIST [*номер строки*] [— [*номер строки*]]
Вывод на печатающее устройство заданного диапазона программных строк.
LOAD *имя файла*[,R]
Загрузка программы из файла с заданным именем и возможное ее выполнение.
MERGE *имя файла*
Объединение программных строк, записанных в файле с заданным именем, со строками, находящимися в памяти.
NAME *старое имя* **AS** *новое имя*
Переименование дискового файла.
NEW
Удаление из памяти всех программных строк, переменных и массивов.
RENUM[[*первый новый номер строки*] [,[*первый старый номер строки*] [, *приращение*]]
Перенумерация программных строк.
RUN [*номер строки*]
Выполнение находящейся в памяти программы; если задан номер строки, то начиная со строки с этим номером.
RUN *имя файла*[,R]
Загрузка и выполнение программы, записанной в файле с заданным именем. Файлы при необходимости остаются открытыми.
SAVE *имя файла*[,A или ,P]
Запись программы в файл.
SYSTEM
Передача управления ДОС ПВМ.
TRON
Переключение в режим трассировки
TROFF
Отмена режима трассировки.

Операторы Бэйсика

BEEP
Включение встроенного звукового сигнального устройства.
CHAIN [**MERGE**] *имя файла*[,[*выражение*][,[**ALL**]] [,**DELETE** *номер первой строки, номер последней строки*]]
Загрузка или объединение программных строк с возможностью предварительного удаления заданного диапазона номеров строк. Выполнение полученной в результате программы, начиная со строки с наименьшим номером или со строки с заданным номером. Возможно сохранение значений переменных.
CIRCLE *(столбец,строка), радиус*[,[*цвет*][,[*начало дуги, конец дуги*] [,*характеристическое отношение*]]]
Вычерчивание окружности, эллипса или дуги.
CLOSE [#][*номер файла/устройства*][,[#][*номер файла/устройства*]] ...
Закрытие файлов и устройств.
CLS

Очистка экрана дисплея.

COLOR [*цвет переднего плана*][,[*цвет фона*][,*цвет границы*]]

Установка цвета фона, переднего плана и границы для экрана в режиме текстового вывода.

COLOR [*цвет фона/границы*][,*палитра*]

Установка цвета фона и границы и выбор палитры переднего плана для режима графического вывода со средней разрешающей способностью.

COMMON *переменная*[,*переменная*] ...

Задание имен тех переменных и целых массивов, значения которых должны сохраняться при выполнении последующего оператора **CHAIN**.

DATA *константа*,[*константа*,] . .

Добавление строковых или числовых констант к списку значений для операторов READ.

DATE$=*строковое значение*

Установка текущей даты для системного календаря.

DEF FN *имя*[*(фиктивная переменная*[,*фиктивная переменная*]...)]=*определение*

Присваивание имени строковой или числовой функции и задание ее определения в виде выражения, в которое могут входить *фиктивные* переменные.

DEF *буква типа* [—*буква*][,*буква*[—*буква*]] ...

Определение типа (INT, SNG, DBL, SNG) для переменных, имена которых начинаются с букв, входящих в один из заданных диапазонов.

DIM *имя массива(индекс* [,*индекс*] ...)[,*имя массива* (*индекс*[,*индекс*] ...)] ...

Распределение памяти для массивов и задание максимального значения индекса по каждой размерности массивов.

DRAW *строка подкоманд*

Вычерчивание фигуры, определяемой заданной *строкой подкоманд*.

END

Завершение программы.

ERASE *имя массива*[,*имя массива*]...

Удаление массивов с заданными именами.

FIELD [#]*номер файла*, *длина* **AS** *строковая переменная*[,*длина* **AS** *строковая переменная*]...

Задание переменных, соответствующих полям файла с прямым доступом.

FOR *переменная-счетчик цикла*=*первое значение* **TO** *последнее значение* [**STEP** *значение приращения*]

Начало цикла FOR/NEXT.

GET [#]*номер файла*[, *номер записи*]

Считывание записи из файла с прямым доступом.

GET (*столбец*1,*строка*1)—(*столбец*2,*строка*2),*имя массива*

Запоминание цвета всех точек заданной прямоугольной области экрана.

GOSUB *номер строки*

Передача управления в подпрограмму строке с *заданным номером*.

GOTO *номер строки*

Передача управления строке с *заданным номером*.

IF *условие* **THEM** *оператор* [:*оператор*]... [**ELSE***оператор*[:*оператор*]...]

Выполнение одного или более операторов в зависимости от условия.

INPUT [;][*"запрос"; или "запрос"*,] *переменная*[,*переменная*] ...

Присваивание всем *переменным* с заданными именами значений, вводимых с клавиатуры. Возможно (но не обязательно) выдавать наводящее сообщение или запрос на ввод с клавиатуры требуемых значений.

INPUT #*номер файла*, *переменная*[,*переменная*]...

Присваивание значений, считываемых из указанного файла или устройства, по очереди всем *переменным* с заданными именами.

KILL *имя файла*

Удаление дискового файла с заданным именем.

[**LET**] *переменная*=*выражение*

Присваивание заданной *переменной* значения *выражения*.

LINE[(*столбец*1,*строка*1)]—(*столбец*2,*строка*2) [,[*цвет*] [,B[F]]]

Вычерчивание на экране дисплея линии или прямоугольника.

LINE INPUT[;][«*запрос*»;] *строковая переменная*

Ввод всех символов, набираемых на клавиатуре до нажатия клавиши ↵ и последующее присваивание их заданной *строковой переменной*. Возможно (хотя и не обязательно) выводить *запрос* на ввод с клавиатуры требуемого значения.

LINE INPUT # *номер файла*,*строковая переменная*

Ввод всех символов из указанного файла, до ближайшего символа возврата каретки, за которым следует символ перехода к следующей строке, а затем присваивание введенных символов *строковой переменной*.

LOCATE [*строка*][,[*столбец*] [,[*видимость*][,[*первый*] [,*последний*]]]]

Перемещение курсора на заданную позицию (определяемую номерами соответствующих строки и столбца). Определение степени видимости курсора на экране, а также его размера и формы.

LPRINT [**USING** *шаблон*;] *список переменных*[;]

Печать на основном системном печатающем устройстве, а в остальном производятся те же действия, что и в случае оператора PRINT.

LSET *переменная поля*= =*строковое значение*

Присваивание заданной *переменной поля* файла с прямым доступом *строкового значения* (начиная с крайней левой позиции поля).

MID$(*строковая переменная, первый символ* [,*количество символов*])=*строковое значение*

Замена части *строковой переменной* (начиная с заданного первого символа) на указанное в команде *количество символов* строкового значения.

NEXT [*переменная-счетчик цикла*][,*переменная-счетчик цикла*]...

Окончание одного или более FOR/NEXT-циклов.

ON ERROR GOTO *номер строки*

Задание *номера программной строки*, которой должно передаваться управление в случае ошибки.

ON *выражение* **GOSUB** *номер строки*[,*номер строки*] ...

Передача управления подпрограмме в *строку с номером*, совпадающим с одним из перечисленных, в зависимости от значения выражения.

ON *выражение* **GOTO** *номер строки*[,*номер строки*]...

Передача управления *строке с номером*, совпадающим с одним из перечисленных, в зависимости от значения *выражения*.

OPEN *имя файла/устройства* [**FOR** *режим*] **AS** [#] *номер файла/устройства* [**LEN**=*длина записи*]

Присваивание *номера файлу/устройству* с заданным именем и задание ре-

жима последсвательного доступа (INPUT, OUTPUT или APPEND), который будет использоваться. Указывая длину записи и опуская конструкцию FOR, можно установить режим прямого доступа.

OPTION BASE *наименьшее значение индекса*

Установка *наименьшего значения* индекса (0 или 1) для всех массивов.

PAINT (*столбец, строка*) [,*цвет-заполнитель* [, *цвет границы*]]

Закрашивание заданным *цветом-заполнителем* области экрана, ограниченной указанным цветом *границы*.

PLAY *строка подкоманд*

Исполнение встроенным звуковым устройством мелодии, определяемой заданной *строкой подкоманд*.

POKE *смещение, значение байта*

Запись по адресу *смещения* заданного *значения байта*.

PRESET (*столбец, строка*)[,*цвет*]

Вывод на экран дисплея отдельной точки; если *цвет* для нее не задан, то используется цвет фона.

PRINT [*список значений*] [;]

Вывод на экран перечисленных в списке значений, если они указаны.

PRINT [#*номер файла/устройства*,] [USING *шаблон*;] *список значений*

Вывод перечисленных в списке значений на экран, либо на другое заданное устройство, либо в заданный файл. Задавая шаблон, можно выводить данные в нужном *формате*.

PSET (*столбец, строка*) [,*цвет*]

Вывод на экран дисплея отдельной точки; если *цвет* для нее не задан, то используется цвет текстовой информации.

PUT#*номер файла*[,*номер записи*]

Занесение записи в файл с прямым доступом. PUT (*столбец,строка*), *имя массива*[,*параметр смеси*]

Воспроизведение на экране цветов всех точек заданной прямоугольной области. Необязательный последний параметр позволяет смешивать заданные цвета точек (представленные в виде массива своих номеров) со стандартными цветами экрана.

RANDOMIZE *целое значение*

Выбор набора случайных чисел по заданному *целому значению*.

READ *переменная*[,*переменная*]...

Присваивание переменным с заданными именами значений из списка, созданного операторамиDATA.

REM [*комментарий*]

Задание комментария в программе; содержащиеся в тексте комментария операторы не выполняются.

RESTORE [*номер строки*]

Восстановление положения указателя списка значений операторовDATA.

RESUME[*номер строки или* **NEXT**]

Возобновление выполнения программы после выявления и обработки ошибки оператором ON ERROR GOTO.

RETURN [*номер строки*]

Возврат управления из подпрограммы оператору, стоящему непосредственно за последним оператором GOSUB (или ON-GOSUB). Возврат управления в строку с *заданным номером* допустим только в расширенном Бэйсике.

RSET*переменная поля*= =*строковое значение*

Заполнение заданного поля файла с прямым доступом указанным строковым значением, выровненным по крайней правой позиции поля.

SCREEN [*режим*][,[*свечение*] [,[*активная страница*] [,*обозреваемая страница*]]]

Выбор режима работы экрана и цвета свечения установка *активной страницы* для операторов вывода и *обозреваемой страницы*, которая будет появляться на экране.

SOUND *частота,длительность*

Генерация тона.

STOP

Прекращение выполнения программы.

SWAP *переменная,переменная*

Обмен значениями между двумя переменными.

TIME$=*строковое значение*

Установка системного счетчика времени.

WEND

Окончание цикла WHILE/WEND.

WHILE *условие*

Начало цикла WHILE/WEND

WIDTH [*номер устройства,*]*длина строки*

Установка *длины* строки для дисплея или какого-либо другого устройства.

WIDTH *имя устройства, длина строки*

Определение длины строки для устройства с заданным именем; указанная длина устанавливается не сразу, а только после разблокирования соответствующего устройства.

Функции в Бэйсике

ABS(*числовое значение*)

Вычисление абсолютной величины *заданного числа*.

ASC(*строковое значение*)

Определение числового кода для первого символа в заданном *строковом значении*.

ATN(*числовое значение*)

Вычисление арктангенса заданного *числового значения*.

CDBL(*числовое значение*)

Преобразование заданного *числового значения* в значение с двойной точностью.

CHR$(*код*)

Определение символа, соответствующего заданному числовому коду.

CINT(*числовое значение*)

Округление заданного *числового значения* до ближайшего целого числа.

COS(*числовое значение*)

Вычисление косинуса заданного *числового значения*.

CSNG(*числовое значение*)

Преобразование заданного *числового значения* в значение с обычной точностью.

CSRLIN

Выдача номера экранной строки, соответствующей текущему положению курсора.

CVD(*строковое значение*)

Преобразование заданного восьмисимвольного *строкового значения* в значение с двойной точностью (функция, обратная по отношению к MKD$).

CVI(*строковое значение*)

Преобразование заданного двухсимвольного *строкового значения* в целое значение (функция, обратная по отношению к MKI$).

CVS(*строковое значение*)

Преобразование заданного четырехсимвольного *строкового значения* в значение с обычной точностью

(функция, обратная по отношению к MKS$).

DATE$

Выдача текущей системной даты.

EOF(*номер файла*)

Выдача значения «Истина» (—1) или «Ложь» (0) в зависимости от того, был ли достигнут конец заданного файла или нет.

ERL

Выдача номера строки, в которой была обнаружена последняя ошибка.

ERR

Выдача кода последней обнаруженной ошибки.

EXP(*числовое значение*)

Возведение константы *e* (2.718282) в степень, равную заданному *числовому значению*.

FIX(*числовое значение*)

Преобразование заданного *числового значения* в целое путем отбрасывания дробной части.

FN*имя* (*значение*[,*значение*]...)

Вызов функции с заданным именем, которая предварительно была определена оператором DEF FN; в этом определении содержится также и информация о количестве и типах значений, которые необходимо задавать при вызове *данной функции*.

FRE(*строковое значение* или *числовое значение*)

Выдача сведений об объеме свободной памяти. Задание *строкового значения* вызывает реорганизацию области памяти, предназначенной для хранения строковых данных.

INKEY$

Выдача информации о том, какая клавиша на клавиатуре нажималась в текущий момент.

INPUT$(*число* [,[#]*номер файла/устройства*])

Считывание заданного числа символов с клавиатуры, какого-либо устройства или файла.

INSTR([*начальный символ*], *исходная строка*, *искомая строка*)

Поиск в исходной строке первого вхождения искомой строки, начиная с заданного начального символа, если он указан, или с самого первого символа исходной строки.

INT(*числовое значение*)

Поиск наибольшего целого числа, не превосходящего заданное числовое значение.

LEFT$(*строковое значение*, *длина*)

Выделение из строкового значения подстроки заданной длины, начиная с крайнего левого символа.

LEN (*строковое значение*)

Определение количества символов в заданном строковом значении.

MID$ (*строковое значение*, *начальный символ* [,*длина*])

Выделение части строкового значения, начинающейся с заданного начального символа. Задавая длину, можно ограничить число символов выделяемой подстроки.

MKD$(*числовое значение*)

Преобразование заданного числового значения в значение с двойной точностью и затем представление его в виде восьмисимвольного строкового значения (функция, обратная по отношению к CVD).

MKI$(*числовое значение*)

Округление заданного числового значения до целого и представление последнего в виде двухсимвольного строкового значения (функция, обратная по отношению к CVI).

MKS$(*числовое значение*)

Преобразование заданного числового значения в значение с обычной точностью и представление последнего в виде четырехсимвольного строкового значения (функция, обратная по отношению к CVS).

PEEK(*смещение*)

Выдача содержимого ячейки памяти по заданному смещению и текущему адресу сегмента.

POINT(*столбец*,*строка*)

Определение цвета точки экрана, соответствующей заданным координатам.

POS(*числовое значение*)

Выдача номера столбца экрана, соответствующего текущему положению курсора; указываемое числовое значение при выполнении функции не используется, но обязательно должно быть задано.

RIGHT$(*строковое значение*,*длина*)

Выделение подстроки заданной длины, соответствующей крайним правым позициям исходного строкового значения.

RND[(*числовое значение*)]

Вычисление случайного числа, заключенного между нулем и единицей.

SGN(*числовое значение*)

Выдача знака заданного числового значения.

SIN(*числовое значение*)

Вычисление синуса заданного числового значения.

SPACE$(*число*)

Генерация заданного *числа* пробелов.

SPC(*число*)

Пропуск заданного *числа* позиций в выводимой на печать строке.

SQR(*числовое значение*)

Вычисление квадратного корня из числового значения.

STRING$(*длина*, *строковое значение* или *код*)

Генерация строки заданной *длины*. Все символы этой строки одинаковы и совпадают с первым символом заданного *строкового значения* либо с символом, имеющим заданный числовой код, в зависимости от того, какой из этих параметров указан.

STR$ (*числовое значение*)

Преобразо-ание заданного *числового значения* в строку символов.

TAB(*столбец*)

Продвижение к заданной позиции в выводимой на печать строке.

TAN(*числовое значение*)

Вычисление тангенса заданного *числового значения*.

TIME$

Выдача системного времени.

VAL(*строковое значение*)

Преобразование *строкового значения* в числовое.

Команды ДОС ПВМ

Версия 1.10

дисковод:

Назначение заданного *дисковода* в качестве используемого по умолчанию, т. е. при выполнении команд с неуказанным в явном виде идентификатором дисковода.

[*дисковод*:]*файл*[*опции*]

Работа с командным или пакетным *файлом*. Количество и вид задаваемых *опций* определяются конкретным командным или пакетным *файлом*.

CHKDSK [*дисковод*:]

Выдача информации о целостности диска и об использовании свободного пространства.

COMP[*дисковод*:]*файл*[*ди-*

сковод:][*файл*]

Сравнение содержимого двух файлов.

COPY [*дисковод:*]*исходный файл* [*опция*] [+[*дисковод:*]*объединяемый файл* [*опция*]] [*дисковод:*][*результирующий файл*][*опция*] [/V]

Копирование *исходного файла* в *результирующий* с возможностью объединения с заданным файлом. Если указан параметр /V, то осуществляется проверка полученной копии. Опция может иметь вид /A (в этом случае исходный файл копируется до первого символа конца файла и этот же символ добавляется в качестве последнего в результирующий файл) или /B (исходный файл копируется целиком, а в конец результирующего файла не добавляется никаких специальных символов).

DATE [*дата*]

Установка системной даты.

DIR [*дисковод:*][*файл*] [/P] [/W]

Вывод всего дискового справочника или некоторой его части.

DISKCOMP [*дисковод:*] [*дисковод:*] [/1]

Сравнение содержимого двух дискетов (при задании /1 производится сравнение только одной стороны дискетов).

DISKCOPY [*исходный дисковод:*] [*целевой дисковод:*] [/1]

Копирование дискета *исходного* дисковода на дискет *целевого* дисковода (символы /1 вызывают только одностороннее копирование).

ERASE [*дисковод:*]*файл*

Удаление файла с заданным именем из дискового справочника.

FORMAT [*дисковод:*] [/S] [/1]

Разметка дискета под формат записей ДОС ПВМ, выделение и отметка на дискете дефектных участков незаполненного справочника. При задании опции /S форматированный дискет назначается системным диском, а при /1 форматирование производится с одной стороны дискета.

MODE LPT: *номер* [*ширина* [,*высота*]]

Возможная установка *длины* строки и ее высоты (только для некоторых моделей) в печатающем устройстве с заданным *номером* (1, 2 или 3).

MODE [*длина строки*] [,*направление*][,T]

Возможная установка *длины* строки экрана и сдвиг изображения с текстовыми данными или без них.

MODE COM*адаптер*: *скорость в бодах* [,*признак четности*[,*слово*[,*стоповая посылка*[,P]]]]

Определяется протокол последовательной передачи данных, который должен использоваться при работе с *адаптерами* № 1 и 2.

MODE LPT*номер*:=COM *адаптер*

Изменяется направление передачи выводимых данных: они направляются не на печатающие устройства с *номерами* 1, 2 или 3, а на адаптер № 1 или № 2.

PAUSE [*комментарий*]

Приостановка обработки, вывод на экран заданного *комментария* с последующим ожиданием нажатия какой-либо клавиши.

REM [*комментарий*]

Вывод на экран комментария.

RENAME [*дисковод:*]*старое имя новое имя*

Изменение старого имени файла на *новое*.

SYS *дисковод*:

Копирование программных файлов ДОС ПВМ, необходимых для превращения заданного диска в системный (на указанном дисководе должен быть установлен диск, размеченный в соответствии с опцией /S).

TIME [*время*]

Установка системного времени.

TYPE [*дисковод:*]*файл*

Вывод на экран содержимого заданного *файла*.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Уважаемый читатель!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., д. 2, изд-во «Мир».

Научное издание

Лон Пул

РАБОТА НА ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ

Научный редактор Т. П. Сапожкова
Младший научный редактор Н. И. Сивилева
Художник В. Е. Карпов
Художественный редактор Н. М. Иванов
Технический редактор Л. П. Бирюкова
Корректор В. С. Соколов

ИБ № 5605

Сдано в набор 18.02.86. Подписано к печати 11.10.86.
Формат 60×90¹/₁₆. Бумага тип. № 1.
Печать высокая. Гарнитура литературная.
Объем 12,00 бум. л. Усл. печ. л. 24,00.
Усл. кр.-отт. 24,00. Уч.-изд. л. 23,63. Изд. №6/4539.
Тираж 80000 экз. Заказ № 2275. Цена 2 руб.

ИЗДАТЕЛЬСТВО «МИР»
129820, ГСП, Москва, И-110, 1-й Рижский пер., 2

Ордена Октябрьской Революции
и ордена Трудового Красного Знамени
МПО «Первая Образцовая типография»
имени А. А. Жданова Союзполиграфпрома
при Государственном комитете СССР
по делам издательств, полиграфии
и книжной торговли. 113054, Москва, Валовая, 28