PROPER BASIC

BRIAN C. WALSH
University of Liverpool

JOHN WILEY & SONS
Chichester • New York • Brisbane • Toronto • Singapore

Б .УОЛШ
ПРОГРАММИРОВАНИЕ НА БЕЙСИКЕ

Перевод с английского И. В. Емелина

МОСКВА «РАДИО И СВЯЗЬ» 1988

ББК 32.973
У 63 УДК 681.3.06:519.682.2
Уолш Б.
У 63 Программирование на Бейсике: Пер. с англ. - М.: Радио и связь, 1988. - 336 с: ил.
ISBN 5-256-00437-9
В книге известного английского автора содержатся сведения о языке программирования Бейсик, предназначенные как для начинающих (основные правила программирования, а также более сложные конструкции языка с примерами их применения при программировании алгоритмов сортировки), так и для опытных программистов (расширение языка для работы с внешними устройствами ЭВМ, технология разработки программ и методы их отладки) . Приведено большое число детально разработанных примеров и задач с решениями.
Для широкого круга программистов.
Редакция переводной литературы
Производственное издание БРАЙАН УОЛШ ПРОГРАММИРОВАНИЕ НА БЕЙСИКЕ
Заведующая редакцией О. В. Толкачева Редактор Е. А. Засядько Художественный редактор Т. В. Бусарова Переплет *художника* О. С. Белова Технический редактор Л. А. Горшкова Корректор Г. Г. Казакова
ИБ № 2083
Подписано в печать с оригинала—макета 21.03.88. Формат 60X90/16. Бумага тип. № 2 Гарнитура Пресс-роман. Печать высокая . Усл. печ. л. 21,0. Усл. кр.-отт. 21,25. Уч.-изд. 20,76. Доп. тираж 100000 экз. Изд. №22806 Зак. № 358 Цена 1 р. 40 к.
Издательство "Радио и связь" 101 000 Москва, Почтамт, а/я 693
Отпечатано в тип. Прейскурантиздата. 125 438, Москва, Пакгаузное шоссе, 1
ISBN 5-256-00437-9 (рус.)
ISBN 0-471-90081-8 (англ.)
Издательство "Радио и связь", 1987

ПРЕДИСЛОВИЕ К РУССКОМУ ИЗДАНИЮ

Язык программирования Бейсик во многом является антиподом современных языков программирования, например языка Паскаль. В отличие от последнего, в Бейсик включались не столько "правильные", сколько "удобные" конструкции и средства. В частности, в нем широко используются разного рода умолчания, что считается совершенно недопустимым в современных языках. В результате процесс написания простой программы на Бейсике для человека, не склонного к педантичности, представляется гораздо более естественным. Это привело к тому, что с появлением относительно недорогих, но мощных микроЭВМ Бейсик по числу своих приверженцев вышел на первое место, оставив позади Паскаль и Фортран.

Как и всякий живой язык, в процессе своего развития Бейсик распался на множество версий, причем сложилось такое положение, что общая часть этих версий предоставляет чересчур ограниченные возможности, и ни одна из версий не стала доминирующей. Поэтому в предлагаемой широкому кругу читателей книге Б. Уолша рассматривается целый спектр версий Бейсика, используемых на ЭВМ самого разного класса — от большой системы ICL2904 до переносной микроЭВМ ZX81. Материал книги подобран так, что для повторения приведенных в ней программ читателю, как правило, не понадобится прибегать к руководствам по имеющейся в его распоряжении вычислительной системе.

Основной тезис Б. Уолша состоит в том, что при правильном подходе к разработке программ влияние присущих Бейсику недостатков может быть существенно снижено. Автор последовательно демонстрирует применение метода разработки по принципу "сверху вниз" на примерах как простых, так и достаточно сложных программ.

Эта книга окажется особенно полезной тем читателям, для которых программирование не является профессией, но дает возможность быстрее и легче получать требуемые результаты. Ее можно рассматривать как своеобразное введение в программирование для непрофессионалов и рекомендовать инженерам и научным работникам, школьникам старших классов, студентам технических вузов.

И. В. Емелин

ПРЕДИСЛОВИЕ

Эта книга рассчитана, с одной стороны, на обучение новичка в области программирования языку Бейсик, а с другой — на ознакомление программиста-практика с основными методами программирования. В конце большинства глав приводятся дополнительные материалы или полный разбор конкретной задачи, что при первом чтении можно пропустить. Они предназначены для более глубокого ознакомления с программированием и ЭВМ. Цель данной книги — научить программированию на Бейсике, а не дать описание этого языка, которое можно найти во многих книгах и руководствах. В настоящей книге основное внимание уделяется процессу разработки программ с целью обеспечить как новичков, так и опытных программистов солидной базой для написания программ. Можно согласиться с тем, что Бейсик — не лучшее средство реализации современных методов разработки, но за счет разумного сочетания этих методов и свойств языка можно добиться
многого.

Элементам языка Бейсик дается критическая оценка; указывается их место в широком контексте развития современных языков программирования.

Демонстрация любого языка программирования на примерах и задачах опирается на конкретную реализацию этого языка. Чтобы вызвать как можно более широкий интерес к данной книге, в ней принят достаточно разумный стандарт; кроме того, приводятся таблицы и примеры программ для нескольких мулътитерминальных больших систем и наиболее популярных
микроЭВМ.

Даются некоторые указания для тех, кто собирается использовать эту книгу в качестве самоучителя. Изучите те главы, которые Вам наиболее необходимы, и пользуйтесь книгой при работе за терминалом ЭВМ. Детали синтаксиса языка Бейсик четко выделены в тексте книги и найти их достаточно легко.

Глава 1 знакомит с электронно-вычислительными машинами и несколькими операторами языка программирования Бейсик. В ней объясняется, какими функциональными устройствами Вам придется пользоваться, и разрабатывается простая программа на Бейсике, которую надо попытаться выполнить на Вашей ЭВМ, возможно, предварительно прочитав гл. 2.

В гл. 2 обсуждаются процедуры пользования терминалом или микроЭВМ и подготовки программы. В ней описано, как пользоваться терминалом или микроЭВМ, как вводить программу и управлять ее работой с помощью команд. Обратите особое внимание на команды LIST и RUN. Ограничьтесь чтением только тех разделов, которые соответствуют Вашей системе

Глава 3 завершает введение в элементарное программирование на Бейсике. После прочтения этой главы Вы будете способны написать множество программ и, возможно, захотите остановиться (по крайней мере, на какое-то время). Убедитесь, что Вы правильно поняли действие оператора IF и организацию циклов с помощью операторов FOR-NEXT. Не жалейте время на выполнение тривиальных примеров, пока не убедитесь, что поняли их. Прежде чем переходить к составлению программ, необходимо убедиться, что Вы хорошо овладели основными понятиями.

Главу 4 надо прочитать тем, кто хочет пользоваться всеми средствами, предоставляемыми языком Бейсик. В ней обсуждаются массивы и обработка строк символов. Для последующего более глубокого чтения в конце главы приводится подробное введение в методы сортировки и поиска.

Чтение гл. 5 необходимо для развития техники разработки программ и собственно программирования. Не стоит и пытаться решать серьезные задачи до тех пор, пока Вы не познакомитесь с назначением функций и подпрограмм и методами их применения, описываемыми в этой главе. Основное назначение главы — подчеркнуть необходимость составления правильного и обдуманного плана любой солидной программы, прежде чем перейти к ее написанию.

В гл. 8 представляются "файлы" и приводится подробный перечень всех типов файлов, доступ к которым возможен средствами языка Бейсик. При первом чтении сосредоточьтесь на использовании программных файлов и простых последовательных файлов. В этой главе описаны особенности многих различных систем, но можно ограничиться тем ее разделом, который наиболее соответствует Вашей ЭВМ.

В гл. 9 полезно заглядывать в любое время, так как в ней подытожены методы разработки программ и исправления ошибок. При первом чтении подразд. 9.2.2 можно опустить, но обратите особое внимание на заключение по методам разработки.

В гл. 6 и 7 описаны расширения языка Бейсик и методы работы с матрицами. Эти главы окажутся полезными более опытным программистам. Книга разделена на четыре части в соответствии с указанным выше распределением материала.

Автор хотел бы поблагодарить профессора Ливерпулского университета Дж. Л, Алти за поддержку и советы при подготовке этой книги, а также за разрешение использовать материалы, включенные в "Самоучитель по программированию на Бейсике", подготовленный в Ливерпулской лаборатории вычислительной техники.

Автор очень благодарен д-ру Р. Крэйвену из Университета графства Эссекс за возможность воспользоваться в гл. 5 его программой STORY, а также д-ру К. Силку из Кильского университета за возможность воспользоваться в гл. 7 его программой GAUSS.

Автор весьма обязан П. Ленгу из Лизерпулского университета за советы и помощь при подготовке рукописи и Р. Саттон за терпеливое приведение рукописи к удобочитаемому виду.

ЧАСТЬ 1 (ВВОДНАЯ)

Этот материал в особенности полезен тем, кто только начинает пользоваться ЭВМ.

1. ВВЕДЕНИЕ

Теперь ЭВМ перестает быть несколько отдаленным и таинственным электронным "волшебным ящиком" и быстро становится привычным элементом повседневной жизни. Широкому распространению ЭВМ в значительной мере способствует появление микроЭВМ, небольших по размерам и очень дешевых в производстве, но тем не менее обладающих всеми атрибутами больших ЭВМ. Если Вы обойдете вычислительный центр, в котором установлена большая ЭВМ, то увидите большие стойки, в которых смонтированы функционирующие компоненты машины. Аналогичные компоненты присутствуют и в микроэлектронных схемах микроЭВМ. Все эти системы, большие и малые, представляют собой ЭВМ общего назначения, требующие наборов команд для выполнения самых разных задач. Эти команды можно задавать разными способами. Один из наиболее распространенных способов — использование языка программирования Бейсик. Его называют языком программирования высокого уровня, так как он позволяет задавать ЭВМ команды с помощью подходящих слов почти как на английском языке и не требует детального знания свойств электронных компонентов ЭВМ. Вообще говоря, отсюда следует, что написанные на Бейсике программы должны быть мобильны, т. е. должны исполняться на любой ЭВМ с Бейсиком. К сожалению, на разных ЭВМ версии Бейсика не вполне идентичны, однако у всех систем существует достаточно большая общая часть, которая в основном и используется в данной книге. Кроме того, в книге излагаются особенности многих основных вариантов Бейсика, но так как язык продолжает развиваться, то невозможно описать все малоизвестные свойства любой версии Бейсика, с которой Вы
можете столкнуться.

Язык Бейсик был разработан в 1963 году профессорами Дартмутского колледжа (США) Кемени и Куртцем. Его основным достоинством считается простота обучения и применения. Словарь языка Бейсик очень ограничен по сравнению с английским языком, но, в отличие от английского языка, им надо пользоваться очень точно. ЭВМ представляет собой "инертный ящик" из электронных компонентов, выполняющий только то, что приказано, — не больше и не меньше. Поэтому все задаваемые ей команды должны быть недвусмысленными. Рис. 1.1 иллюстрирует очень общие свойства языка, подобного английскому, требования, предъявляемые ЭВМ к машинному языку, и связующую роль промежуточного языка, подобного Бейсику.

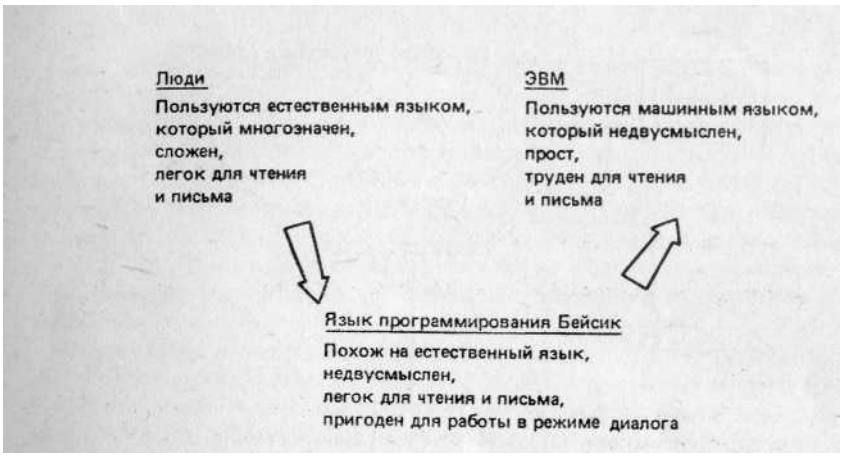


Рис. 1.1. Язык программирования служит посредником между людьми и ЭВМ

Написанные на языке Бейсик команды еще не находятся в той конечной форме, в которой они могут быть восприняты вычислительными устройствами. Они должны быть переведены в команды машинного кода довольно сложной программой, именуемой компилятором или интерпретатором. Такой компилятор обычно поставляется вместе с ЭВМ и является основной частью того, что предстает перед Вами как "система с Бейсиком". Те свойства языка, которые компилятор способен

воспринимать, определяют форму доступной Вам системы. Вообще говоря, чем меньше ЭВМ, тем более скромные возможности будут предоставлены системой.

1.1. ЭВМ

Язык программирования Бейсик рассчитан на работу с ЭВМ в режиме диалога. На рис. 1.2 изображен обычный набор компонентов, составляющих основу системы с Бейсиком.

Устройством ввода обычно служит клавиатура. С ее помощью машине передаются инструкции, команды и информация. *Устройством вывода* обычно является экран видеомонитора, на котором изображается текстовая и графическая информация. Наряду с результатами вычислений на экране видеомонитора изображаются символы, набираемые на клавиатуре. Вместе два этих прибора образуют *видеотерминальное устройство* (ВТУ) .

К большой вычислительной системе может быть подключено много ВТУ и других устройств ввода и вывода, которые могут работать одновременно. Когда сразу много пользователей обращается с помощью ВТУ к системе с языком Бейсик, ЭВМ (а точнее, находящаяся в ней специализированная программа, называемая *операционной системой)* хранит плоды труда каждого пользователя отдельно и таким образом, что каждому из них кажется, будто он работает на машине один.

МикроЭВМ очень похожа на ВТУ, поскольку ее активные логические эле-

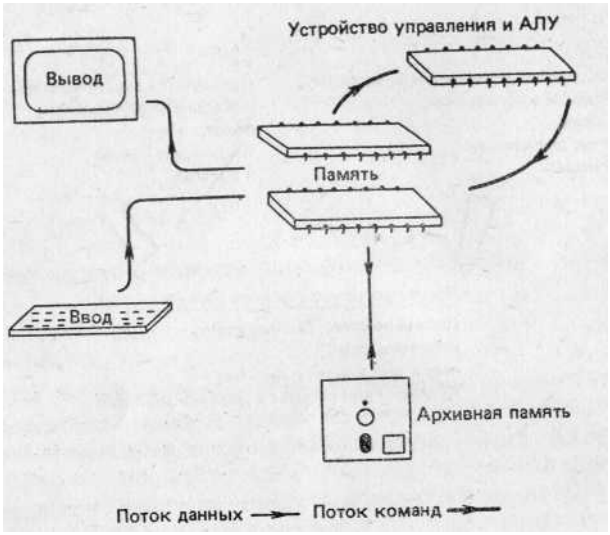


Рис. 1.2. Основные компоненты ЭВМ и общая схема потоков данных

менты настолько малы, что без труда могут поместиться внутри клавиатуры или кожуха экрана.

Активные логические элементы образуют наиболее интересную часть ЭВМ; в больших машинах она называется *центральным процессором* и выглядит как отдельный ящик. Однако в микроЭВМ она вместе с некоторыми другими элементами может оказаться одной микросхемой. Наиболее важной частью центрального процессора является *арифметико-логическое устройство* (АЛУ), работа которого напоминает работу карманного калькулятора тем, что производит сложения, вычитания, умножения и логические операции над любыми данными, которые могут быть предоставлены.

Данные, предоставляемые АЛУ, и команды, указывающие, как эти данные должны быть обработаны, хранятся в *памяти* ЭВМ. Память представляет собой область, сохраняющую команды и данные, попадающие в нее с клавиатуры ВТУ или другого устройства ввода. Центральный процессор читает данные из памяти и записывает данные в память. Результаты любых вычислений копируются из памяти на устройство вывода, например на принтер или экран ВТУ. Память используется только как рабочая область; она не рассчитана на долговременное хранение команд и данных.

Программой называют набор команд, манипулирующих данными. Ограничившись помещением программы в память и включением АЛУ, многого не добьетесь. Требуется что-то такое, что могло бы подавать данные в АЛУ в соответствии с командой программы. Для этого используется *управляющее устройство.* Оно интерпретирует в единицу времени по одной команде, сообщает АЛУ, что надо делать, и поставляет АЛУ необходимые для каждой ко-

манды данные. Действия управляющего устройства определяются своеобразным "сердцебиением" ЭВМ, порождаемым *тактовым генератором.* Каждый тактовый импульс генератора вызывает элементарное действие управляющего устройства, и быстродействие всей системы определяется частотой тактов. Для микроЭВМ типичное значение тактовой частоты равно 4 000 000 тактов в секунду. В результате этого ЭВМ, снабженная программами, работающими всего лишь несколько секунд, может показаться нам весьма сложной и даже "умной". На самом деле ее работа состоит из выполнения достаточно простых операций с очень большой скоростью, значение которой было указано выше.

Память ЭВМ не сохраняет команды и данные надолго. В действительности содержимое памяти большинства ЭВМ теряется после выключения питания. В памяти хранится текущая программа, которая исполняется или готовится к исполнению. После завершения работы она уничтожается, и вместо нее можно ввести новую программу. Надолго программы и данные можно сохранить в *архивной памяти,* из которой они могут быть скопированы в память ЭВМ и в которую они могут быть скопированы из памяти ЭВМ. Архивная память обычно представляет собой кассету или бобину с магнитной лентой или магнитные диски. Диски бывают разные: очень дорогие жесткие с большой емкостью, обычно способные хранить десятки миллионов символов, и дешевые гибкие, способные хранить сотни тысяч символов.

1.2. К ПРОСТОМУ БЕЙСИКУ

Перед тем как ЭВМ сможет начать работу над задачей, команды и данные должны быть размещены в определенных ячейках памяти ЭВМ. Каждой ячейке приписан определенный машинный адрес, подобный номеру дома на улице. Вместо того чтобы оперировать этими таинственными машинными адресами, язык Бейсик предоставляет удобный способ пометки как команд, так и данных. Каждая команда, которую можно ассоциировать с предложением на английском языке, помечается номером, присваиваемым Вами, программистом, а каждый элемент данных адресуется по имени (обычно состоящему из одной буквы) , которое также выбирается Вами. Таким образом, команда

10 LET A = 3

хранится в ячейке памяти, на которую можно ссылаться по номеру 1 0, и вызывает запоминание числа 3 в ячейке памяти по имени А. По команде 20 LET B=A+2

из ячейки по имени А будет извлечено содержимое, в данном случае 3, к нему будет добавлено 2 и результат (5) будет помещен в ячейку данных, указанную в левой части. Таким образом,

30 LET B=A означает B←A,

т. е. значение, полученное из ячейки данных А, копируется в ячейку денных В. Содержимое ячейки А не изменяется командой и остается тем же, что и до исполнения команды, но содержимое В перезаписывается на новое, взятое из А.

Действия, подобные перемещению значений из одной ячейки данных в другую (А или В в приведенном выше примере) , применимы и к командам. Каждая команда программы на Бейсике идентифицируется своим номером. Одно из назначений этих номеров, идентифицирующих строку программы, — указать порядок обработки строк. Программа на Бейсике состоит из последовательности команд, выполняемых друг за другом. Система с Бейсиком всегда начинает исполнение программы со строки с наименьшим номером и обрабатывает по одной строке за прием до тех пор, пока не будет обнаружен конец программы. Программа очень похожа на кулинарный рецепт: для получения успешного результата каждое действие должно быть проделано в нужном порядке.

Обсуждавшиеся выше ячейки данных А и В называются *переменными* ввиду того, что такие ячейки содержат значения, которые могут изменяться при исполнении программы.

Учтите, что система с Бейсиком не выполняет строки команд сразу же после ввода их в ЭВМ, а ждет, пока Вы не предложите ей это сделать, задав команду RUN. Пусть, к примеру, в ЭВМ были введены следующие строки:

```
10 A = 1
20B = -4
30C=A + B
```

Тогда после ввода в ЭВМ команды RUN переменной А будет дано значение 1, затем переменной В — значение —4 и в заключение переменной С — значение 1-4, равное —3.

1.2.1. ПЕРВАЯ ПРОГРАММА

Ниже приведен полный текст программы:

```
10 INPUT A
20 LET B=A+2
30 PRINT B
40 END
```

При исполнении программы ЭВМ обрабатывает ее, начиная со строки с наименьшим номером (10), до строки с наибольшим номером (40) . Первая команда, INPUT, заставляет ЭВМ ждать до тех пор, пока на клавиатуре ВТУ не будет набрано какое-либо число. Это число будет дано ячейке А в качестве значения, а затем ячейке В будет дано значение А, увеличенное на 2. Значение В изображается на дисплее ВТУ, и исполнение программы завершается.

При вводе в ЭВМ каждый оператор Бейсика набирается отдельной строкой и предваряется номером. Операторы выполняются в порядке возрастания номеров, что не обязательно совпадает с порядком ввода операторов.

Например, приведенную выше программу с равным успехом можно было бы записать и так:

```
10 INPUT A
40 END
30 PRINT B
20 LET B=A+2
```

Не обязательно, но полезно давать операторам номера с регулярным приращением 10: если при вводе какая-то строка оказалась пропущенной, то ей можно дать промежуточный номер и она займет требуемую позицию. Если приведенная выше программа содержится в памяти и мы наберем

25 LET B=B+10 то получим следующий результат:

```
10 INPUT A
20 LET B = A+2
25 LET B = B+10
30 PRINT B
40 END
```

Номера операторов. Для микроЭВМ номер оператора — число от 0 до 65 535, которое указывается перед каждым оператором программы на Бейсике. Для многих больших ЭВМ номер оператора — число от 1 до 9999.

Следующая программа будет исполняться в точности так же, как и та, что приводилась в начале этого раздела:

```
1 INPUT A
29 LET B = A+2
1021 PRINT B
9998 END
```

Раздел 2.2 содержит сведения о том, как набирать программу и как запускать ее. Можно попробовать проделать эти процедуры перед тем, как несколько подробнее познакомиться с использованными выше операторами.

1.2.2. ОПЕРАТОР INPUT

Оператор INPUT A вызовет приостанов программы, ожидание набора на клавиатуре числа и копирование этого числа в переменную (или ячейку данных) A, после чего исполнение программы продолжится.

В операторе INPUT можно указывать только имена переменных. Например, оператор

290 INPUT K записан правильно, а оператор

290 INPUT K+1 записан неправильно.

Оператор INPUT

Общая форма записи:

INPUT *элемент 1, элемент 2, элемент 3,...*

Оператор INPUT *обеспечивает ввод данных в программу с терминала. Исполнение программы приостанавливается, на ВТУ изображается знак вопроса (или что-нибудь в этом роде), и система ожидает ввода значений для элементов, перечисленных списком в тексте оператора. Значения должны быть разделены запятыми, и тип значения должен совпадать с типом соответствующей ему переменной.*

Если данных введено меньше, чем ожидалось, то система будет повторять приглашение к вводу значений до тех пор, пока все элементы списка не получат значения.

Одним оператором можно ввести несколько чисел, например:

10 INPUT A, B, Z

Всякий раз, когда программе требуется дополнительная информация, на экране изображается в какой-либо форме приглашение к вводу; обычно им служит знак вопроса (?). ЭВМ воспринимает набираемые данные только после нажатия на клавишу возврата каретки. В случае ввода многих данных можно указывать в одной строке несколько значений данных, разделяя их запятыми. Если значений указано меньше, чем требуется данных, то ЭВМ снова повторит приглашение к вводу. Например, при выполнении приведенного выше оператора 10 значения данных можно было бы задавать следующим образом (ответы пользователя подчеркнуты) :

? 10,50 (нажатие клавиши возврата каретки)

?—2 (нажатие клавиши возврата каретки)

или

?10 (нажатие клавиши возврата каретки)

?50, -2 (нажатие клавиши возврата каретки) В обоих случаях окажется, что A содержит 10, B 50, а Z - 2.

1.2.3. ОПЕРАТОР PRINT

Оператор PRINT B вызовет изображение (или печать) содержимого переменной B на экране ВТУ. В отличие от оператора INPUT, в оператор PRINT можно включать достаточно сложные выражения. Эти выражения вычисляются, и результат вычислений печатается. Например, если B содержит 5, то в результате выполнения оператора

20 PRINT B+10

будет изображено 15.

С помощью одного оператора PRINT можно выводить несколько чисел.

Например,

30 PRINT B, Z, A

изобразит значения трех элементов данных на одной строке.

В операторе PRINT кроется много возможностей. Накопив опыт, с помощью этого оператора можно получать самые разнообразные выходные формы в том числе графические изображения с низкой разрешающей способностью. Другие свойства оператора PRINT обсуждаются в подразд. 3.5.1.

1.2.4. ПРИСВАИВАНИЕ. ОПЕРАТОР LET

Оператор LET B=A*10 берет значение переменной A, умножает его на 10 и запоминает в переменной B. Учтите, что содержимое A не изменяется. Основные возможности, предоставляемые оператором LET, приводятся в табл. 1.1.

Если требуется произвести умножение, то обязательно надо указывать оператор *. Оператор LET B=10A не допускается; надо писать LET B = 10* A или LET B = A* 10. Ваша ЭВМ может воспринять и оператор вида LET B =A10, но при этом результатом будет не перемножение значений, а копирование содержимого ячейки по имени A10 (см. имена переменных, подразд. 1.2.6).

Числа, предоставляемые программе в качестве данных или появляющиеся в операторах программы, как в приведенных выше примерах операторов LET, могут быть положительными, нулем или отрицательными и при этом *целыми,* например 0, -7, 5, 2167, или *вещественными* (десятичными) числами, например 22.1,-0.6,449.305.

В случае вещественных чисел очень большие или очень маленькие значения иногда гораздо удобнее выражать в виде произведения на десять в степени целого числа (табл. 1.2). Оператор PRINT автоматически выводит эти большие и маленькие числа в формате с умножением на десять в степени целого числа.

Таблица 1.1

Действие	Операция	Пример
Сложение	+	LET C = A + B
Вычитание	–	LET C = A – 1Ø.5
Умножение	*	LET Z = A *5
Деление	/	LET T = Z/A
Возведение в степень	↑, или ^, или **	LET B = A ↑ 3
Копирование		LET B = A
Копирование с обращением знака	–	LET B = –A

Таблица 1.2

Число	Запись в виде произведения	Запись для ЭВМ
5ØØØØØ	$5 \cdot 1Ø^{5}$	5.ØE+5 или 5E5
46 927	$4.6927 \cdot 1Ø^{4}$	4.6927E+4 или 4.6927E4
Ø.ØØØ42	$4.2 \cdot 1Ø^{-4}$	4.2E–4
–31.27	$-3.127 \cdot 1Ø^{1}$	–3.127E–1

Учтите, что форматы 5E5, 50E4 и 0.5E6 эквивалентны между собой и представляют число 500000.

У Вашей ЭВМ существует достаточно практичный предел значений экспоненты (показателя степени при десяти, равного значению, указываемому справа от E). Учтите, что экспонента должна быть целым числом.

1.2.5. АРИФМЕТИЧЕСКИЕ ВЫРАЖЕНИЯ

В одном операторе LET можно выполнять несколько арифметических операций:

```
10LETA = 3.14159*R↑2
10LETV = A*B + C/2.1
10LETX= (Y+22.7)/180
```

В Бейсике существует строго определенный порядок выполнения арифметических операций (правила старшинства), в достаточной степени соответствующий алгебраическим правилам, и его недопонимание может приводить к странным на первый взгляд результатам. Эти правила старшинства взяты не произвольным образом, а с таким расчетом, чтобы гарантировать один, и только один результат у любого выражения, входящего в состав оператора Бейсика. Старшинство операций определяется следующим образом:

а) в первую очередь вычисляются части выражения, заключенные в скобки;

б) затем производятся операции возведения в степень;
в) затем выполняются умножения и деления;
г) последними производятся сложения и вычитания.
Эти правила дополняются правилом, согласно которому в случае, если два соседних действия равны по старшинству, вычисления производятся слева направо.
Предположим, что в примерах табл. 1.3 B = 100, C = 50и D = 20. Содержащиеся в ней выражения на Бейсике сконструированы с помощью приведенных выше правил так, что они эквивалентны алгебраическим формулам левой колонки.
Если бы правил старшинства не было, то выражение примера 6 можно было бы интерпретировать как (100*50-20)/50 = 4980/50 =99.6, что не совпадает с правильным результатом 4999.6, полученным выше. Помните, что система с Бейсиком всегда следует этим правилам, так что и Вам, программисту, всегда надо им следовать. Эти правила позволяют записать приведенный выше пример 4 в форме B/C/D, что эквивалентно (B/C)/D = (100/50)/20 = 2/20 = 0.1 и приводит к правильному результату, но такая запись имеет несколько "обескураживающий вид", поэтому ее лучше избегать.
Будьте внимательны со скобками. Пример 3 нельзя записать в виде B+C/D=100+(50/20)=102.5, что соответствует алгебраической формуле b+c/d, а не тому, что требуется. Аналогично, если бы пример 7 был записан в виде B + C↑10/D, то результатом было бы B+(C↑10)/D = 100+ (50↑10)/20 - большое число, являющееся результатом $b + c^{10}/d$, а не заданной формулы.

Таблица 1.3

Алгебраическая формула	Выражение на Бейсике	Результат (при B = 1ø, C = 5ø, D = 2ø)
1. b + c – d	B+C–D	(1ø+5ø) – 2ø = 13ø
2. b – c – d	B–C–D	(1ø–5ø) – 2ø = 3ø
3. $\frac{b+c}{d}$	(B + C) /D	(1ø+5ø) /2ø = 7.5
4. $\frac{b}{cd}$	B/ (C*D)	1ø / (5ø*2ø) = ø.1
5. $b^{d/1ø}$	B↑ (D/1ø)	1ø↑ (2ø/1ø) = 1ø↑2 = 1øøø
6. $bc-\frac{d}{c}$	B*C–D/C	(1ø*5ø) – (2ø/5ø) = 4999.6
7. $b+c^{1ø/d}$	B + C↑ (1ø/D)	(1ø+5ø↑ (1ø/2ø)) = 1ø+5ø↑ø.5 = =1ø7.ø71ø7

Оператор LET
Общая форма записи:
LET *переменная = выражение*
"Переменная" - имя переменной (ячейки данных), которая может хранить целые или вещественные числа либо строки символов.
"Выражение " дает результат того же типа (целый, вещественный, строка символов), что и у переменной в левой части.
Оператор LET *присваивает переменной результат вычисления выражения. Все элементы данных, входящие в состав выражения в правой части, должны к моменту вычисления иметь определенное значение. Другими словами, они должны предварительно получить значение с помощью операторов* INPUT, LET *или* READ *{последний обсуждается в разд. 4.4).*
Для большинства систем служебное слово LET *можно опускать.*
Применение оператора LET в случае, когда по обе стороны от знака присваивания (=) указана одна и та же переменная, может вызвать недоумение, если Вы не вполне усвоили тот факт, что правая часть вычисляется до того, как результат копируется в левую часть. Оператор
LET N=N+1
возьмет текущее значение переменной N, добавит к нему 1 и затем скопирует полученный результат обратно в N; Если N первоначально содержала 10, то в результате вычислений в N попадет 11. Операторы такого вида широко используются в программах на Бейсике, так как их действие состоит

в добавлении 1 к N (в приращении значения N) при каждом исполнении оператора. В этом случае N действует как счетчик. Аналогично допускаются операторы вида
LET A=A*10
17

В данном случае первоначальное значение A умножается на 10 и результат возвращается в A.
Обычно служебное слово LET в таких операторах разрешается опускать, что и будет делаться в последующих главах этой книги. Употребление слова LET обязательно только на персональной ЭВМ (ПЭВМ) ZX81 фирмы Sinclair.

1.2.6. ПЕРЕМЕННЫЕ И ИМЕНА

До сих пор мы обсуждали переменные, помеченные однобуквенным именем. Эти переменные являются основными элементами, которыми манипулируют команды программы; они получают данные, предоставляют данные для обработки и получают измененные данные в ходе исполнения программы.
Во всех версиях Бейсика переменным можно давать однобуквенные имена, а также имена, состоящие из буквы и следующей за ней цифры от 0 до 1. Ниже приводятся примеры допустимых имен переменных:
A, B, N, A0, Z9, M6, D2
Таким образом, всего допускается 26 + 26 * 10 = 286 имен для ячеек данных, отводимых под переменные. Следующие имена недопустимы:
5C (начинается с цифры, а не буквы) A/ (содержит операцию)
Некоторые версии Бейсика позволяют давать переменным более длинные имена; ниже перечисляются основные варианты.
Имена переменных
Большинство систем с Бейсиком для микроЭВМ
Одна буква (от A до Z) или буква, за которой следует цифра (от 0 до 9).
Большинство систем с Бейсиком для микроЭВМ
Одна буква, буква и цифра, две буквы.
Бейсик Microsoft
Все, что выше, и в дополнение - буквенно-цифровые имена с произвольной длиной. Однако идентификация имени производится только по первым двум буквам. Таким образом, имена TOTAL и TOTE допустимы, но описывают ту же переменную, что и TO.
Бейсик BBC
Одна буква или буква, за которой следует любое число букв или цифр. Все они имеют значение. Таким образом, имена TAX и TAXIDERMIST допустимы и соответствуют разным переменным.
Единственно правилен подход, принятый в Бейсике BBC. Имена переменных должны быть способны выражать ту величину, которую они представляют. Тогда программа становится более удобочитаемой и менее вероятны ошибки при разработке программ. Легко представить, к каким серьезным ошибкам может привести ситуация, когда допускаются длинные имена переменных, но при этом для различения имен используются только два символа. При работе с системой такого рода безопаснее всего ограничиваться именами только из двух букв.
18

Наконец, в некоторых версиях Бейсика, допускающих длинные имена, не разрешается использовать в их составе имена операторов Бейсика (это не относится к Бейсику BBC, если только имя переменной не начинается со служебного слова).
Выбор имени всецело в ведении программиста, но одного только тщательного выбора имени недостаточно: переменным должны даваться значения до того, как этими переменными начинают пользоваться. Некоторые версии Бейсика могут привить Вам ложное ощущение безопасности,

автоматически полагая перед началом работы программы значения всех переменных равными 0 (см. команды CLR, CLEAR в приложении III). С этой особенностью надо обходиться осторожно, так как при ее отсутствии в программу могут проникать трудно обнаруживаемые ошибки.

Обратите внимание на то, что в данной книге используются минимальные соглашения о допустимых именах переменных, т. е. одна буква плюс одна цифра. Это гарантирует, что приведенные в книге примеры программ смогут работать на большинстве систем. Другими словами, эти программы мобильны или близки к тому, чтобы быть как можно мобильнее. Если Вы собираетесь передавать свои программы на другие ЭВМ, то стоит придерживаться этой практики.

Прежде чем закончить обсуждение переменных, надо затронуть еще один вопрос: какие значения могут содержаться в переменных? Память большинства ЭВМ состоит из групп по 8 бит в каждой.(Каждый бит может содержать только значения 0 или 1, а комбинация из восьми битов позволяет представлять числа от 0 до 255.) Эти 8 бит называются байтом, и емкость памяти ЭВМ обычно измеряют в тысячах байтов. При этом употребляется аббревиатура К, например 10К байт. В контексте ЭВМ 1К = 1024, так что память емкостью 10К в действительности содержит 10 240 байт.

В Вашей системе с Бейсиком под каждую переменную выделяется определенное число байтов памяти. Переменные бывают как минимум двух типов: один из них позволяет хранить числа с определенной точностью (шесть или семь десятичных цифр), а другой - символы. Переменные последнего типа называются *строковыми,* так как могут содержать строку символов. Например, A1 может содержать значение 1, или 1.237, или 5E+10 и так далее, а переменная A$ (знак $ служит признаком строкового типа) может содержать букву B или строку HELLO THERE. Пока что можете не слишком заботиться о строковых переменных; они будут детально обсуждаться позже в разд. 4.2.

Во многих системах с Бейсиком для микроЭВМ по причинам экономичности, точности, а также для удобства программирования вводятся дополнительные типы переменных. Они могут быть короче обычных переменных (для экономии памяти) и содержать только целые числа или быть длиннее обычных переменных и обеспечивать большую точность в критичных местах численных методов. Не обращайте на них внимания до тех пор, пока не

сможете вполне компетентно писать работающие программы. Ради полноты изложения приведем основные типы переменных:

Числовой тип, A (стандартный; используется для вещественных значений)	Вещественные числа, имеющие 6-7 значащих цифр.. Допустимый диапазон экспоненциальной части 10^{-38} ... . . 10^{+38}. Обычно требуется по 4 байта памяти на каждую переменную этого типа. В стандарте Бейсика BBC 9 значащих цифр и 5 байт на переменную
Строковый тип, A$ (стандартный; признаком строковой переменной служит знак $после имени)	Строка символов. Может содержать от 0 до 255 символов; для размещения в памяти переменной этого типа требуется по одному байту на каждый символ ее значения
Целый тип, A% (признаком целого типа служит знак % после имени)	Целые числа в диапазоне ± 32767. Для их хранения обычно требуется по 2 байта на каждое число. В Бейсике BBC допустимый диапазон значений $\pm 200 \cdot 10^6$, а занимаемая память 4 байта на число
Вещественный тип с двойной точностью, A# (признаком этого типа служит знак # после имени)	Вещественные числа; подобны числам стандартного вещественного типа, но имеют 16-17 значащих цифр. Одной из немногих версий, имеющих этот тип, является Бейсик Microsoft

1.2.7. ОПЕРАТОР END

Желательно, чтобы программа завершалась оператором END и чтобы его номер был в программе наибольшим, так что этот оператор будет исполняться последним. Разумный и аккуратный подход к программированию требует безоговорочного выполнения этого правила, поскольку большинство систем с Бейсиком позволяют опускать оператор END, но в некоторых системах считается ошибкой его отсутствие и они не будут выполнять программу до тех пор, пока этот оператор не будет вставлен.

Оператор END

Общая форма записи:

END

Оператор вызывает завершение исполнения программы. Обычно является последним оператором программы.

Некоторые системы с Бейсиком, в основном для микроЭВМ, не требуют I *обязательного присутствия оператора* END.

1.2.8. ФОРМАТ ТЕКСТА ПРОГРАММЫ

Каждый оператор программы записывается с новой строки, в начале которой указывается его номер. Некоторые системы с Бейсиком для микроЭВМ позволяют указывать несколько операторов в одной строке; операторы разделяются двоеточием (:), и номер указывается только для первого оператора, например:

```
10 LET A = 1 : LET A =A + 1 :PRINT A
20
```

Порядок исполнения операторов в этом случае — слева направо. Вообще говоря, указание нескольких операторов в одной строке не рекомендуется. Однако в некоторых системах за счет этого может экономиться память, и, может быть, Вам придется воспользоваться такой формой записи, если при нормальном расположении операторов по одному в строке для Вашей программы не хватило объема памяти. Позднее мы увидим, что с некоторыми ограничениями запись нескольких операторов в одной строке может применяться при структурировании программы.

При обработке каждого оператора машина обычно игнорирует все пробелы (это не относится к строкам символов) . Таким образом, операторы

```
20 LETB=A + C
```

и

```
20 LET B=A + C
```

скорее всего, окажутся в равной степени приемлемыми. Однако разумное использование пробелов обычно облегчает ввод программы. ЭВМ может хранить введенную программу в компактной внутренней форме, исключив все пробелы, и добавлять их для приведения текста программы к некоторому стандартному виду при распечатке программы на ВТУ. Удаление пробелов может вызвать серьезные огорчения, если пробелы тщательно вставлялись в текст программы с тем, чтобы выделить вложенные структуры.

1.3. ЧТЕНИЕ ПРОГРАММЫ

Одним из важнейших качеств, которые потребуются от Вас как от программиста, является умение "читать" программы и получать определенное представление об их функциях. Вы должны уметь распознать "куски" программы и определить их назначение, не сбиваясь из-за неудачного выбора имен переменных или формата текста программы. По мере накопления опыта Вы обнаружите, что после некоторой модификации эти куски программ можно использовать для Ваших целей. В качестве отправной точки этого процесса рассмотрим следующую программу на Бейсике, в которой используются представленные в этой главе операторы. Она вычисляет площадь и длину окружности круга по значению его радиуса. (Площадь равна πr^2, а длина окружности $2\pi r$, где r-радиус, π=3. 14159 ...)

```
10 LET P=3.14159
20 INPUT R
```

```
30 LET A=P*R*R
40 LET C=2*P*R
50 PRINT R,A,C
60 END
```

Напомним, что первое число в каждой строке является номером оператора и обязательно должно быть указано и что операторы исполняются в порядке возрастания из номеров, в нашем случае — с 10 до 60.

Когда ЭВМ получает команду на исполнение этой программы (см. команду RUN в разд. 2.4), то переменной P присваивается значение 3.14159. Затем программа приостанавливается для ввода с клавиатуры значения переменной

R, далее продолжает работу и вычисляет по значениям этих переменных значения A и C. Наконец, на ВТУ изображаются значение R (радиус) и вычисленные величины A (площадь) и C (длина окружности).

Программа остается в памяти ЭВМ и может быть повторно исполнена для вычислений при другом значении R. Таким образом, она является программой общего назначения, получающей с клавиатуры важное для работы значение переменной (радиуса). Обратите внимание на то, что мы не хотим каждый раз набирать значение константы я. Поэтому оно устанавливается в программе. Конечно, можно было бы не вводить значение переменной R, а присваивать его в программе, но в этом случае программу надо было бы изменять всякий раз, когда требуется произвести вычисления при другом значении радиуса. В нашем случае сделать это нетрудно, но при более сложных программах это не так-то просто, и, кроме того, мы потеряли бы общность, которую так нелегко получить.

Структура этой программы очень проста, что бывает нередко и в случае очень больших программ. Она такова:

Ввод

Обработка

Вывод

Таким образом, вначале вводятся значения, затем следует обработка данных, после чего производится вывод результатов. Может показаться само собой разумеющимся, что такой схемы и надо придерживаться. Однако дистанция между постановкой задачи и написанием программы для ее решения столь велика, что в практику программирования приходится включать средства преодоления этой дистанции. Описанный выше подход к разработке программы является одним из таких средств. Вы обнаружите, что последовательность ввод—обработка—вывод применима к множеству программ и может служить полезной отправной точкой при разработке Вашей программы.

В заключение этой главы бросим еще один взгляд на программу и обсудим новый оператор REM. Этот оператор позволяет Вам включить в текст программы любые комментарии. Комментарии игнорируются машиной и предназначены помочь человеку (программисту). Например,

```
10 REM ЭТОТ ОПЕРАТОР - КОММЕНТАРИЙ 20
REM И ЭТОТ ТОЖЕ
30 REMARK ВНОВЬ КОММЕНТАРИЙ
```

Оператор REM

Общая форма записи;

REM *комментарии*

В комментариях допустим любой текст, в том числе и служебные слова Бейсика. Оператор REM *используется для добавления комментариев к распечатке программы и игнорируется при исполнении программы.*

Надежды возлагаются на то, что оператор REM в сочетании с приданием программе четкой структуры сделает программу самодокументирующейся. Иначе говоря, программисту будет достаточно иметь только распечатку про-

граммы и он сможет без привлечения дополнительной информации точно установить, что эта программа делает и как ее исполнить. К сожалению, эта цель редко достигается; тем не менее она стоит того, чтобы к ней стремиться, так как в результате получаются программы, которые легче модифицировать и развивать.

При исполнении приведенной выше программы выдается три числа в одной строке. Было бы полезно указывать, что эти числа означают. Это можно сделать с помощью оператора PRINT в строке 46, изображающего названия, которые указываются в нем заключенными в кавычки:

```
5 REM ПЕРВАЯ ПРОГРАММА
10 LET P=3.14159
15 REM
20 INPUT R
25 REM
30 LET A=P*R*R
40 LET C=2*P*R
45 REM
46 PRINT "РАДИУС"."ПЛОЩАДЬ"."ОКРУЖНОСТЬ"
50 PRINT R,A,C
60 END
```

Для разъяснения программы желательно включать в нее операторы REM, однако они занимают место в памяти, и если память ограничена, то Вам может показаться, что их лучше избегать. Тем не менее включайте их всюду, где только возможно.

Учтите, что версия Бейсика BBC отличается от большинства других способом позиционирования чисел и строк символов при выводе их оператором PRINT. При работе с ней указанные выше заголовки РАДИУС, ПЛОЩАДЬ, ОКРУЖНОСТЬ окажутся сдвинутыми по отношению к изображенным под ними числам. Один из путей добиться правильного соответствия — добавить к каждому заголовку по несколько пробелов и тем самым сдвинуть их вправо. Таким образом, надо воспользоваться оператором

```
46 PRINT" РАДИУС"," ПЛОЩАДЬ"," ОКРУЖНОСТЬ"
```

Если Вы пользуетесь Бейсиком BBC, то запомните это на будущее при исполнении других примеров книги.

1.4. ПРИМЕРЫ ПРОГРАММ

Первая из приведенных ниже программ преобразует галлоны в литры. Количество галлонов запоминается в переменной A и преобразуется с помощью соотношения 1 галлон = 4,54 л (ответы, набираемые на клавиатуре пользователем, подчеркнуты).

```
10 REM ПРЕОБРАЗОВАНИЕ ИЗ ГАЛЛОНОВ В ЛИТРЫ
20 LET A=7
30 LET B=A*4.54
40 PRINT B
50 END
RUN (команда исполнения программы)
31.78 (результат)
 END AT LINE 50 (сообщение системы о завершении исполнения программы)
```


Выданный ответ говорит, что 31.78 л эквивалентны 7 галлонам. Обратите внимание на выдаваемое системой заключительное сообщение END AT LINE 50; оно может быть иным в другой системе с Бейсиком. Некоторые системы в конце исполнения программы выдают OK, но в книге будет использоваться первая форма.

Чтобы сделать работу с программой более удобной, добавим в нее несколько операторов PRINT и заменим строку 20 на оператор INPUT. При этом в ЭВМ потребуется ввести только новые строки. Полный текст новой программы станет следующим:

```
10 REM ПРЕОБРАЗОВАНИЕ ИЗ ГАЛЛОНОВ В ЛИТРЫ
18 PRINT "ВВЕДИТЕ ЧИСЛО ГАЛЛОНОВ"
20 INPUT A
30 LET B=A*4.54
35 PRINT "ГАЛЛОНЫ"."ЛИТРЫ"
40 PRINT A,B
50 END
RUN
ВВЕДИТЕ ЧИСЛО ГАЛЛОНОВ
? 7 (программа запрашивает ввод и получает его)
ГАЛЛОНЫ ЛИТРЫ
7 31.78 END AT LINE 50
```

8 заключение приведем достаточно сложную программу, в которой операторы PRINT изображают текст и ответы в одной строке и которая использует в строке 50 новый тип переменной.

Эта программа вычисляет заработную плату (G), получая за счет ввода количество отработанных за неделю часов (H), почасовую ставку (R) и фамилию работника. (Ответы пользователя подчеркнуты.)

```
10 REM ПРОГРАММА НАЧИСЛЕНИЯ ЗАРПЛАТЫ
20 PRINT "ВВЕДИТЕ ОТРАБОТАННОЕ ВРЕМЯ И ПОЧАСОВУЮ СТАВКУ"
30 INPUT H.R
40 PRINT "ВВЕДИТЕ ФАМИЛИЮ"
50 INPUT N$
60 LET G=H*R
70 PRINT N$;" ПОЛУЧИТ";G;"РУБ."
80 END
RUN
ВВЕДИТЕ ОТРАБОТАННОЕ ВРЕМЯ И ПОЧАСОВУЮ СТАВКУ
ВВЕДИТЕ ФАМИЛИЮ
?Тихонов
ТИХОНОВ ПОЛУЧИТ 50 РУБ END AT LINE 80
```

Обратите внимание на применение строковой переменной N$ для хранения фамилии работника. Подобные переменные содержат текст в виде строк символов. Точки с запятой в операторах PRINT, как показано выше, дают иной эффект, чем запятые. В деталях это будет обсуждаться позже, а пока что попробуйте сами найти разницу.

УПРАЖНЕНИЯ

(Служебное слово LET в операторах присваивания опускается.)

1.1. Какие конечные значения примут переменные после использования следующих фрагментов программ?

(а) 10 A*2 20 B=A 30 Z=A+2*B 40 A=B

(б) 10 A=-5 20 B=-A
30 Q=1/(A+2)-B

(в) 10 B=0
20 A=(B+1)/(2+B) 30 Z=-2/(A*(B-A

(г) 10 A=0
20 A=A+10
30 A=A*A 40 A=A+10

1.2. Напишите операторы присваивания (LET), вычисляющие по следующим формулам:

(а) $a=b+c^2$

(б)a =(a+b)/c,
*(в)*y=(x+a)/(y-b),
1.3. Что делает следующая программа?

```
10 A=22
20 B=-4.1
30 D=7.6
40 C=109.34
50 Z=A+B+C+D
60 Z=Z/4
70 PRINT Z
80 END
```

1.4. Перепишите программу из упражнения 1.3, заменив операторы 10 - 40 на один или несколько операторов INPUT.
1.5. Требуется написать финансовую программу, начисляющую простой процент с ДАННОЙ суммы в течение данного времени. Пусть переменная M содержит сумму денег,
время в годах, а R - процентная ставка (в пределах 0... 100). Соответствующая формула такова:
процент = M*T*(R/100)
(.Указание: возьмите для программы стандартную форму ввод-обработка-вывод.)

2. ПОДГОТОВКА И ИСПОЛНЕНИЕ ПРОГРАММЫ

Прежде чем приступить к созданию программы, надо включить питание вычислительной системы и привести ее в состояние готовности к принятию символов, набираемых на клавиатуре. Для этого, возможно, придется выполнять несколько операций, а каких именно — зависит от типа используемого
Вами оборудования.
В следующем разделе описаны основные возможности различных систем, а также указаны причины, заставляющие их функционировать именно так, а не иначе.

2.1. ПОДГОТОВКА К ВВОДУ

Вообще говоря, вычислительные системы можно подразделить на два типа: с одним терминалом, рассчитанные на обслуживание одного пользователя, и мультитерминальные, рассчитанные на обслуживание многих пользователей. Для систем каждого типа характерна схожесть основных операций; в последующих разделах описываются возможности, предоставляемые системами каждого типа.
Большинство микроЭВМ рассчитано на работу только с одним пользователем, что существенно упрощает выполняемые ими действия. Для этой группы систем существует два подхода к реализации языка Бейсик (а равно и других языков программирования). Они описаны в подразд. 2.1.1 и 2.1.2.1
2.1.1. ОДНОПОЛЬЗОВАТЕЛЬСКИЕ СИСТЕМЫ (СИСТЕМЫ С БЕЙСИКОМ В ПЗУ) Проще всего встроить систему с Бейсиком в постоянное запоминающее устройство (ПЗУ). ПЗУ допускает только чтение данных и организовано в виде группы байтов точно таким же образом, как и обычная память. Однако при изготовлении в ячейки ПЗУ записывается система с Бейсиком. Содержимое ячеек фиксируется, и его можно читать, но нельзя изменить. Для сохранения информации в ПЗУ электропитания не требуется, и эту память можно представлять себе как содержащую "замороженные" программы. В ПЗУ можно хранить любые программы, а система с Бейсиком и является довольно сложной программой или группой программ. Скажем, игры с ЭВМ обеспечиваются игровыми программами, хранящимися в ПЗУ. Такие ЭВМ должны иметь и обычную память, о которой уже говорилось в гл. 1. Эту память называют запоминающим устройством с произвольной выборкой (ЗУПВ), потому что к любой ее части (не обязательно к группе смежных байтов) возможен доступ как для чтения, так и для записи.
Когда включается питание этих простых систем, часть системы с Бейсиком автоматически копируется из ПЗУ в ЗУПВ, и система готова к немедленной работе. Ниже описана типичная процедура начала работы (то, что набирается пользователем на клавиатуре, подчеркнуто) ;
(а) Включите питание; выключатель обычно находится на задней стенке корпуса ЭВМ. При этом может произойти небольшая задержка; в некоторых системах в процессе считывания системы с Бейсиком из ПЗУ экран может заполняться символами.

(б) На экране появится сообщение типа

```
***HAPPY DAYS INC. BASIC VERSION 2.0***
32767 BYTES FREE
READY
```

(в) После этого можно вводить любую команду или инструкцию Бейсика. Ниже приведена часть сеанса работы с системой; чтобы помочь Вам повторить его, в скобках в правой части даются пояснения. Клавиша возврата каретки может быть маркирована RETURN, CR, ACCEPT или NEWLINE.

PRINT"HI THERE"	(наберите эту строку и нажмите клавишу возврата каретки)
HI THERE	(реакция на оператор PRINT при работе в режиме немедленного исполнения, см. разд. 2.4)
10 INPUT A	(нажмите клавишу возврата каретки)
20 INPUT B	(нажмите клавишу возврата каретки)

30C = A + B	(нажмите клавишу возврата каретки)
40PRINT A; "+"; B; "="; C	(нажмите клавишу возврата каретки)
50 END	
RUN	
?2	(нажмите клавишу возврата каретки)
?27.3	(нажмите клавишу возврата каретки)
2 + 27.3 = 29.3	(выводится результат работы программы)

В сообщении, выдаваемом в момент начала работы системы, обычно указывается число свободных байтов памяти (ЗУПВ) . Это максимальное число, которым Вы располагаете для своей программы. Что касается набранной выше программы, то при исполнении операторов 10 и 20 она запрашивает ввод посредством изображения вопросительного знака (?). Учтите, что любой текст, заключенный в кавычки в операторе PRINT, выводится в точности таким, как он указан в операторе.

У вычислительной системы с Бейсиком в ПЗУ много преимуществ: с ней очень легко начать работу; не требуются ни диск, ни магнитофон; кроме того, система с Бейсиком может иметь много средств, разработанных с учетом особенностей конкретной аппаратной части ЭВМ.

2.1.2. ОДНОПОЛЬЗОВАТЕЛЬСКИЕ СИСТЕМЫ (СИСТЕМЫ С БЕЙСИКОМ НА ДИСКЕ)

Один из очень гибких способов организации работы ЭВМ состоит в том, чтобы снабдить ее определенной операционной системой (ОС) . Она представляет собой программу, которая управляет работой аппаратной части ЭВМ, берет на себя все заботы об адресах, связанных с аппаратурой, и предоставляет возможность управлять работой ЭВМ с помощью ряда простых команд. частности, ОС позволяет достаточно простым способом загрузить систему Бейсиком или любую другую программу либо систему с языком програм-

мирования и запустить ее в работу. При исполнении других систем, по крайней мере, часть ОС остается в памяти ЭВМ. Следовательно такая ЭВМ может быть в трех различных состояниях:

1. "Голое" состояние без ОС или какой-либо другой программы. Обычно ЭВМ в таком состоянии способна воспринять только простую команду загрузки в память операционной системы, чаще всего с диска.
2. Состояние после загрузки ОС, когда можно задавать много команд, манипулирующих программами и данными. Одной из таких команд может быть команда загрузки системы с языком программирования, например с Бейсиком.
3. В третьем состоянии (после загрузки системы с Бейсиком) можно вводить операторы или команды Бейсика. При необходимости можно завершить работу системы с Бейсиком и вернуться к взаимодействию с операционной системой.

Одна из подобных операционных систем получила очень широкое распространение. Эта ОС выпускается фирмой Digital Research и называется CP/M. Она представляет очень удобную среду для разработки программ. Так как сама CP/M адаптируется к особенностям аппаратной части конкретной ЭВМ, то с точки зрения программ представляемая ею среда выглядит одинаково и не зависит от ЭВМ. Ниже описан пример сеанса работы с ЭВМ под управлением операционной системы CP/M (команды, набираемые пользователем на клавиатуре, подчеркнуты):

(а) Включается питание ЭВМ (выключатель обычно находится на задней стенке ее корпуса), дисковода, если он не встроен в ЭВМ, и принтера, если таковой имеется.

(б) Реакция ЭВМ на включение питания минимальна: в левом верхнем углу ВТУ появится одиночный символ, например H:

(в) Системная дискета, на которой записаны CP/M и Бейсик, вставляется в дисковод, маркированный буквой A. Обычно это ближний к экрану дисковод; дискета вставляется так, чтобы ее вырез был

направлен к задней стенке корпуса, а наклейка на дискете была обращена к дверце дисковода. Дверца после этого защелкивается. Если есть и другие дисководы, то им будут приписаны имена B и C.

(г) Затем операционная система загружается с дискеты в память; это называется "вызовом" системы. Для вызова системы наберите на клавиатуре букву B и нажмите клавишу возврата каретки; машина среагирует на это, допечатав на экране буквы так, что образуется слово BOOT (вызов) . Сигнальная лампочка на дисководе A должна замигать, что указывает на его активность, и на экране должно появиться сообщение о начале работы системы:

H:B_OOT (нажмите клавишу возврата каретки)
64K HAPPY DAYS INC. VERSION CP/M 2.0 A>

(д) Теперь CP/M загружена; приглашение к вводу (>) показывает, что можно давать команды. Буква A перед знаком > показывает, что в данный

момент активен дисковод A. Любая из команд CP/M будет применяться к файлам на дискете, установленной в дисководе A.

(е) Если произошли отклонения от указанной выше последовательности событий, то нажмите клавишу RESET на верхнем регистре (при нажатой клавише SHIFT) для возвращения в состояние (б) и далее повторите все заново. Сверьтесь с руководством по Вашей ЭВМ: клавиша, нажимаемая на шаге (г), может быть иной для Вашей ЭВМ.

(ж) К настоящему моменту загружена лишь ОС и Бейсик пока что недоступен. Чтобы познакомиться с примером команды операционной системы CP/M, наберите DIR, после чего будет изображен список файлов, находящихся на текущем дисководе (дисководе A) :

A > DIR (нажмите клавишу возврата каретки)
(появится список имен файлов)
A>

(з) Обратите внимание на файл по имени MBASIC.COM; это версия системы с Бейсиком Microsoft. Наберите имя MBASIC для загрузки этой системы:

A > MBASIC (нажмите клавишу возврата каретки)
BASIC80REL. 5.21
(CP/M VERSION)
COPYRIGHT BY MICROSOFT
CREATED 12-DEC-83
30715 BYTES FREE
OK

(и) После ввода команды MBASIC сигнальная лампочка дисковода A должна замигать, и после небольшой задержки должно появиться сообщение о начале работы, аналогичное уже упоминавшемуся выше. Единственным приглашением к вводу со стороны Бейсика является мигающий курсор в первом столбце экрана.

(к) Теперь все готово для сеанса работы с Бейсиком. Попробуйте повторить тот, что описан в подразд. 2.1.1 как часть (в) процедуры начала работы.

При работе с Бейсиком под управлением CP/M Вы не будете замечать наличие операционной системы и сможете исполнять, сохранять и распечатывать свои программы на Бейсике так, как это было описано в гл. 1. При желании можно прекратить работу с Бейсиком, набрав команду SYSTEM, и вернуться к работе с ОС CP/M. Тем самым Вы окажетесь в состоянии (д) описанной выше процедуры и сможете под управлением CP/M создавать, редактировать, перемещать, уничтожать, распечатывать и копировать любые файлы, какие только пожелаете. Если программа на Бейсике не была сохранена в течение сеанса работы с Бейсиком, то при возвращении к CP/M она будет утрачена. Перечень команд CP/M приводится в подразд. 8.5.2, а обсуждение свойств CP/M с Бейсиком включено в гл. 8.

2.1.3. МУЛЬТИТЕРМИНАЛЬНЫЕ СИСТЕМЫ

ЭВМ, способная обслуживать много терминалов, может быть расположена близко или далеко от них, может быть маленькой или очень большой. В этом случае Вам достаточно уметь обращаться с терминалом, а запускать систему
должны специалисты.

Как и у некоторых вычислительных систем, ранее описанных в этой главе, здесь тоже имеется операционная система. Она должна обеспечивать возможность работы ЭВМ с многими пользователями. Одна из ее задач — хранить плоды труда каждого пользователя отдельно от других. Подобная ОС должна работать таким образом, чтобы у Вас создавалось впечатление, что других пользователей у системы нет.

Терминал (или ВТУ) обычно постоянно подключен к ЭВМ, но в тот момент, когда Вам потребуется им воспользоваться, он может быть неактивен. Первым делом надо активизировать терминал, так как вычислительная система не тратит время на проверку состояния тех терминалов, которые считает неактивными. Процедура активизации выглядит следующим образом:

(а) Проверьте, включено ли питание ВТУ.

(б) Дайте ЭВМ знать, что терминал требует ответа. Это осуществляется нажатием специальной клавиши или некоторой комбинации клавиш. Ниже приводятся некоторые из возможных вариантов:
нажмите несколько раз клавишу RETURN или SEND,
или
нажмите клавишу управляющего регистра CONTROL и, удерживая ее
в нажатом состоянии, несколько раз нажмите клавишу A,
или
нажмите клавишу CONTROL и, удерживая ее в нажатом состоянии,
несколько раз нажмите клавишу C.

(в) Система может отреагировать переводом курсора на новую строку, или каким-либо приглашением к вводу или выдаст сообщение о начале обслуживания терминала.

(г) В начале сеанса работы необходимо представиться ЭВМ в качестве правомочного пользователя. Для этого обычно вводятся команды LOGIN или HELLO; они меняются от системы к системе, но всегда включают в себя код, идентифицирующий пользователя, и, возможно, пароль. После этого Вам будет дозволено наряду с другими пользователями работать с системой, о чем обычно сигнализирует появление на экране сообщения, в котором указаны Ваши идентификатор, тип сеанса и другие детали.

(д) Если система специализирована для работы с Бейсиком, то Вы можете' продолжить сеанс так, как это было описано на шаге (в) в подразд. 2.1.1 В противном случае Вам понадобится загрузить в память систему с Бейсиком, обычно набрав простую команду, например BASIC.

Проделав эти манипуляции, Вы сможете вводить и исполнять описанные в этой книге программы. Перед завершением сеанса надо позаботиться о том, чтобы все программы и данные, находящиеся в памяти и требующие сохранения, были запомнены в соответствующих файлах.

В мультитерминальных системах Вы можете делать все что угодно со своими файлами: уничтожать их, копировать, редактировать. Однако доступ к файлам других пользователей обычно можно получить только с их разрешения, для чего предусмотрены специальные команды.

В заключение сеанса работы с Бейсиком не забудьте сообщить операционной системе о завершении работы. Если Вам пришлось загружать Бейсик в память, то сначала надо завершить работу с Бейсиком. Этой цели служат такие команды Бейсика, как QUIT или BYE. В некоторых операционных системах это воспринимается и как сигнал о завершении работы, но в других вслед за этим надо ввести команду, например LOGOUT. Ниже приводится типичная реакция операционной системы на команду завершения работы:
идентификатор пользователя LOGGED OUT AT время, дата
TIME USED = время, затраченное на сеанс.

2.2. ВВОД ПРОГРАММЫ НА БЕЙСИКЕ

После выполнения Вами одной из описанных выше процедур система готова к вводу программы.

Наберите первую строку, не забыв указать номер оператора, и нажмите клавишу возврата каретки (на Вашей клавиатуре она может иметь маркировку RETURN, CR или ACCEPT). Тем самым управление будет передано ЭВМ, которая запомнит введенную строку и выдаст приглашение к вводу следующей строки. Некоторые вычислительные системы, в основном большие мультитерминальные ЭВМ, проверяют правильность строки перед тем, как воспринять ее. Это свойство очень полезно, так как сразу выявляются нередко случающиеся опечатки и такие ошибки, как недостающие скобки. В этом случае система не запомнит ошибочную строку. Большинство систем с Бейсиком для микроЭВМ не обладает таким свойством; они выполняют аналогичные синтаксические проверки только в момент исполнения программы.

Наберите следующую строку, завершив ее ввод нажатием клавиши возврата каретки, и продолжайте до тех пор, пока не введете всю программу.

Ошибки в операторах неизбежны. Процесс изменения операторов называется редактированием. Проще всего изменить текущую строку; это называется редактированием строки.

2.2.1. РЕДАКТИРОВАНИЕ СТРОКИ

Существует два способа исправления ошибок в отдельной строке. Рассмотрим ошибочный оператор

10 INPUS A

(а) Если Вы уже послали эту строку в ЭВМ (нажатием на клавишу возврата каретки), то для исправления ошибки вся строка попросту набирается заново, включая ее номер:

10 INPUT A нажмите клавишу возврата каретки

31

Обычно строку можно удалить, набрав строку из пробелов с тем же самым номером:

10 нажмите клавишу возврата каретки

Набор такой строки вызовет удаление строки 10. Зачастую имеется команда удаления строк по одной или целой группой.

(б) Если клавиша возврата каретки еще не нажата, то можно аннулировать всю строку или попытаться исправить ошибку. Если аннулировать строку проще, то обычно требуется одновременно нажать две клавиши: одна из них — клавиша управляющего регистра (маркированная CTRL или CONTROL), а другая зависит от конкретной системы. Приведем несколько примеров:

ЭВМ 1CL 2903/4 — CONTROL и X

SINCLAIR ZX81 — EDIT и затем NEWLINE

Бейсик Microsoft — CONTROL и X

Если ошибка замечена вскоре после того, как она была сделана, нажмите клавишу удаления символа (маркированную DELETE, или ←, или RUBOUT, или подчеркиванием (_)) столько раз, сколько символов надо удалить. Тем самым уничтожаются символы, которые должны быть посланы ЭВМ. В некоторых системах изображение удаляемых символов может остаться на экране ВТУ. Например, ввод строки

10 INPUS ←T A нажмите клавишу возврата каретки

эквивалент вводу 10 INPUT A. Здесь нажатие на клавишу удаления показано как ← ; при этом удаляется набранный перед ней символ. Чтобы сделать правильной строку

10 INPT A

нажмите клавишу удаления трижды (для уничтожения последовательности символов "T пробел A") :

10 INPT A ←←← UT A нажмите клавишу возврата каретки Запомните, что последней версией данной строки является та, которая находится в памяти и будет взята при исполнении программы. Строки можно вводить в любом порядке; пропущенные строки можно добавить в любой момент до исполнения программы.

Большинство микроЭВМ обеспечивает возможность изображения на ВТУ всей программы или только ее части и перемещения курсора по экрану в процессе изменения программы. Этот способ редактирования называется редактированием экрана и в значительной степени зависит от конкретной

системы. В системе с Бейсиком BBC требуется переместить курсор к изменяемой строке и затем использовать клавишу COPY для создания новой строки. После завершения коррекции строки надо нажать клавишу возврата каретки.

2.3. ПРИМЕРЫ ПРОГРАММ

Ниже представлены для исполнения три программы. После каждой приводится подробное обсуждение ее свойств. При этом потребуются команды

RUN, исполняющая программу, и LIST, изображающая программу на ВТУ. Эти команды подробно описаны в разд. 2.4.

Первая программа выполняет простое преобразование из английских мер длины (футы и дюймы) в метрические:

```
10 REM ПРОГРАММА ПРЕОБРАЗОВАНИЯ
20 PRINT "ЗАДАЙТЕ ДЛИНУ В ФУТАХ И ДЮЙМАХ"
30 INPUT F,I
40   I=I+F*12
50   C=I*2.540
60  C=C/100
70 PRINT "МЕТРИЧЕСКИЙ ЭКВИВАЛЕНТ РАВЕН";C;"M" 80 END
```

Строка 10 служит комментарием и не вызывает никаких действий.

Строка 20 выдает сообщение "ЗАДАЙТЕ ДЛИНУ В ФУТАХ И ДЮЙМАХ".

Строка 30 требует ввода двух элементов данных; укажите их значения вместе в одной строке, разделяя запятыми, или отдельно в разных строках.

Строка 40 преобразует футы в дюймы и добавляет полученный результат к ранее введенному числу дюймов.

Строка 50 преобразует длину из дюймов в сантиметры (считая, что в одном дюйме 2.540 см) .

Строка 60 делением на 100 преобразует длину из сантиметров в метры.

Строка 70 изображает результат в виде текста "МЕТРИЧЕСКИЙ ЭКВИВАЛЕНТ РАВЕН XXX.XX МЕТРОВ".

Раздельное изображение числа метров и сантиметров при выводе результата требует несколько более подробных сведений о Бейсике, чем до сих пор изложено. Необходимые детали можно почерпнуть из обсуждения функции INT в гл. 3.

Строки (или любые символы) можно использовать наряду с числами, но определенные детали их применения несколько отличаются в разных системах с Бейсиком. Следующий пример должен успешно исполняться на большинстве ЭВМ:

```
10 REM ИЛЛЮСТРАЦИЯ РАБОТЫ СО СТРОКАМИ СИМВОЛОВ
20 PRINT "ПОЖАЛУЙСТА, ВВЕДИТЕ ВАШУ ФАМИЛИЮ"
30 INPUT A$
40 PRINT "ПОЖАЛУЙСТА. ВВЕДИТЕ ВАШЕ ИМЯ"
50 INPUT B$
60 C$=B$+" "+A$
70 PRINT "ДОБРЫЙ ДЕНЬ. ";C$;" КАК ПОЖИВАЕТЕ?"
80 END
```

Строка 20 выдает изображение "ПОЖАЛУЙСТА, ВВЕДИТЕ ВАШУ ФАМИЛИЮ".

Строка 30 требует ввода нескольких символов в качестве значения переменной A$.

Строки 40 и 50 имеют аналогичное назначение для переменной B$.

В строке 60 формируется строка текста, состоящая из имени, за которым следует вначале пробел, а затем фамилия. Хотя при этом и используется знак сложения, эта операция называется конкатенацией и представляет собой

слияние строк для образования более длинной строки, в нашем случае запоминаемой в переменной C$.
В некоторых системах для обозначения конкатенации используется не знак сложения, а знак &.
Последняя программа похожа по структуре как на первые две, так и на те, что мы видели ранее, однако она содержит несколько ошибок. Она должна вычислить и изобразить на экране среднее значение трех величин, которые программа получает при вводе. Если программа набрана следующим образом:

```
10 REM ПРОГРАММА ВЫЧИСЛЕНИЯ СРЕДНЕГО ЗНАЧЕНИЯ
20 PRINT "ВВЕДИТЕ 3 ЧИСЛА, ПОЖАЛУЙСТА"
30 INPUT A,B1,C2
40 A1=A+B+C/3
50 PRINT "СРЕДНЕЕ ИЗ";A;B;C;"РАВНО";A0
60 END
```

то исполняться она будет, но при этом будут выдаваться ошибочные результаты.
Запустите эту программу, задавая тестовые значения для трех переменных. Проверьте результат; если он ошибочен, займитесь "отладкой" программы, другими словами, поиском и удалением ошибок. Умение отлаживать программу - искусство, развиваемое практикой. Но если Вы только начинаете заниматься отладкой, полезно задаться двумя вопросами. Во-первых, по какому пути идет выполнение программы (в какой последовательности исполняются операторы, т. е. как передается управление от оператора к оператору); во-вторых, какие значения принимают внутренние переменные в различных точках программы?
Просмотрите программу с помощью команды LIST. Изучение программы показывает, что она исполняется последовательно, без каких-либо отклонений**,** начиная со строки 10 и до строки 60. Затем проверьте значения переменных. Конечные значения переменных можно посмотреть с помощью немедленно выполняемого оператора PRINT (например, PRINT A2, A, B, C). Для изображения промежуточных значений в программу можно вставлять дополнительные операторы PRINT, давая им новые номера строк, например 25,35 и т. д. Исполняйте программу несколько раз, до тех пор пока не добьетесь желаемого результата.

2.4. КОМАНДЫ

Команды представляют собой инструкции, которые даются непосредственно системе с Бейсиком, управляющей работой ЭВМ (на некоторых машинах может дополнительно существовать операционная система, например CP/M, которой пересылаются команды). Команды отличаются от операторов языка Бейсик отсутствием номеров строк.
Запомните, что образующие программу операторы имеют номера строк и исполняются только тогда, когда программа запущена. Команды выполняются немедленно и предназначаются для управления ресурсами системы. На самом деле многие из операторов языка Бейсик можно указывать без номе-

Команда	Назначение	Ссылка
RUN	Исполнение находящейся в памяти программы	Подразд. **2.4.1**
Останов программы	Иногда необходима для прекращения исполнения программы с помощью специальной команды "прерывание", набираемой на клавиатуре	Подразд. **2.4.1**
LIST	Распечатка всех операторов находящихся в памяти программы	Подразд. **2.4.2**
NEW или SCRATCH	Освобождает память, чтобы можно было набирать новую программу	Подразд. **2.4.3**
OLD или GET	Копирует ранее запомненную программу из архивной памяти в память	Подразд. **2.4.4**

	ЭВМ	
SAVE	Сохранение находящейся в памяти	Подразд.
Поиск имен запомненных программ	текущей программы посредством копирования в архивную память В некоторых системах хранится предметный указатель программ,	**2.4.5**
HELLO или LOGIN	запомненных в архивной памяти. Его называют каталогом или справочником.	
SYSTEM	Команда поиска машинно-зависима.	
BYE, или QUIT, или LOGOUT	Мультитерминальные системы требуют, чтобы перед началом работы пользователь сообщил свой код (иногда называемый именем пользователя). Кроме того, может требоваться еще и пароль Только для CP/M: эта команда завершает сеанс работы с Бейсиком и возвращает управление операционной системе CP/M Эта команда завершает сеанс работы с Бейсиком, начатый командой HELLO или LOGIN	

ра строки, и в этом случае они выполняются немедленно. Поэтому четкого различия между операторами и командами не существует. Команда

PRINT 10*32 нажмите клавишу возврата каретки

указанная без номера строки, служит примером оператора, который должен быть отработан немедленно, в результате чего на ВТУ должен появиться результат 320 (см. обсуждение применения этого оператора для отладки в подразд. 9.1.2).

Часть из наиболее употребительных команд обсуждается в последующих разделах. В табл. 2.1 приводится перечень этих команд и вкратце указывается их назначение.

2.4.1. КОМАНДА RUN

Программа загружается в память ЭВМ из архивной памяти или набирается на клавиатуре. Для ее исполнения достаточно набрать команду RUN нажмите клавишу возврата каретки

После получения этой команды система с Бейсиком проводит полную проверку программы и, если ошибок не обнаружено, начинает выполнять операторы по порядку. Рассмотрим процесс исполнения примера законченной программы, приведенного в разд. 1.3:

```
5 REM ПЕРВАЯ ПРОГРАММА
10 LET P=3.14159
15 REM
20 INPUT R
25 REM                                    ,
30   LET A=P*R*B
40   LET C=2*P*R 45 REM
46 PRINT "РАДИУС","ПЛОЩАДЬ","ОКРУЖНОСТЬ"
50 PRINT R,A,C
60 END
```

Пусть эта программа уже набрана и теперь ее можно исполнить (набранная пользователем информация подчеркнута): RUN нажмите клавишу возврата каретки

?2.4 нажмите клавишу возврата каретки (программа запрашивает и получает значение для R)

РАДИУС ПЛОЩАДЬ ОКРУЖНОСТЬ

2.4 18.0956 15.0796
END AT LINE 60 (в некоторых системах предусмотрена выдача такого сообщения, показывающего, на какой строке завершилось исполнение программы)
Эту программу можно исполнять повторно, указывая другие значения R, если давать команду RUN и данные (как показано выше) столько раз, сколько потребуется Команда RUN *Общая форма записи:* RUN *номер строки*
Эта команда предлагает системе начать исполнение находящейся в памяти программы со строки с наименьшим номером. Если в команде указан номер строки (например, RUN 100*), то исполнение программы начнется с указанной строки, а не со строки с наименьшим номером.* В конце подразд. 2.4.4 см. описание расширенной команды RUN для Бейсика Microsoft.
Остановить программу после того, как она начала исполняться, может оказаться труднее, чем запустить ее. Большинство систем позволяет давать команду "приостанов" или "прерывание", останавливающую исполнение программы. Эта команда останавливает выполнение и других функций, например распечатывание программы командой LIST. Но будьте осторожны: большинство микроЭВМ имеет также команду для того, чтобы начать работу системы заново. Если такая команда будет дана вместо команды приостанова, то Вы лишитесь всех программ, находившихся к этому моменту в памя-

ти. Ниже приведены некоторые примеры для различных систем; в случае сомнения обратитесь к руководству по Вашей ЭВМ.

ЭВМ ICL 2903/4	(CONTROL) и
Бейсик Microsoft	A
Прекращение работы программы	(CONTROL) и C
Приостанов работы программы	(CONTROL) и S
Продолжение работы после приостанова	(CONTROL) и O
Персональная ЭВМ ZX81 фирмы Sinclair	(SHIFT) и A
Останов ожидающей ввода программы	
Приостанов исполняемой программы	нажатие пробела
Персональная ЭВМ PET фирмы Commodore	STOP
Персональная ЭВМ BBC	ESCAPE

2.4.2. КОМАНДА LIST
Вероятно, это наиболее часто используемая команда. Она изображает на ВТУ операторы находящейся в памяти программы. Программа может быть законченной или незаконченной. Во время ввода программы полезно просматривать ранее набранные операторы, чтобы иметь ясное представление о том, что именно было воспринято системой, принимая во внимание редактирование, ошибки и т. д. Набранная сама по себе команда
LIST нажмите клавишу возврата каретки
распечатает на ВТУ все операторы находящейся в памяти программы. Методы вывода текста программ, у которых строк больше, чем их помещается на экране, различаются. Некоторые системы разбивают вывод на страницы. Иначе говоря, они изображают 15 или 20 строк, а затем ждут, пока не будет нажата клавиша продолжения. После этого изображаются строки следующей страницы и т. д. Клавиша продолжения может быть маркирована стрелкой (↑). В других системах изображение перемещается вверх при добавлении новой строки внизу экрана. В некоторых системах это происходит очень быстро и существует клавиша для замедления смены изображения, например клавиша REVS у микроЭВМ CBM PET. В других системах надо приостанавливать выдачу изображения (например, ОС CP/M использует CONTROL и S), просматривать изображение и

разрешать продолжение его выдачи (CP/M для возобновления выдачи использует повторное нажатие CONTROL и S).
Все системы с Бейсиком позволяют изображать избранные строки или группы строк. Для этого в команде LIST указываются номера первой и последней из изображаемых строк: по команде
LIST 100-110
на экране будут изображены строки с 100-й по 110-ю; команда
LIST 100-изобразит на экране строки программы с 100-й по последнюю. В некоторых
системах в качестве разделителя границ интервала номеров строк вместо дефиса (-) используется запятая (,), например:
LIST 90,110

Команда **LIST** *Общая форма записи:* LIST *нижний предел - верхний предел или*
LIST *нижний предел, верхний предел*
"Нижний предел " и "верхний предел" ~ числа, устанавливающие минимальное и максимальное значения номеров изображаемых строк. Указывать числа не обязательно.
Команда LIST *изображает построчно текст текущей программы, находящейся в памяти. Изображение начинается с нулевой строки или со строки, номер которой не превышает нижнего предела, и заканчивается последней строкой программы или строкой с ближайшим к верхнему пределу номером.*
Способ изображения, постраничный или с построчным движением, определяет режим распечатывания программы командой LIST, но может определять и режим вывода результатов при исполнении программы по команде RUN, а также режим ввода операторов и команд. Система с Бейсиком BBC начинает работу в режиме построчного движения изображения и может быть переведена в режим постраничного изображения одновременным нажатием на клавиши CTRL и N. Для изображения "новой" страницы надо нажать клавишу SHIFT. Нажатие на клавиши CTRL и Q вызывает обратный переход от режима постраничного изображения к режиму построчного движения.
2.4.3. КОМАНДА NEW (ИЛИ SCRATCH, ИЛИ UNSAVE)
Команда NEW освобождает рабочую память от любой программы или операторов, которые в ней содержатся. Она освобождает память, чтобы можно было набирать новую программу. Если по команде LIST распечатываются какие-либо строки, то они составляют текущую программу. Команда NEW уничтожит эти строки; если ввести после нее команду LIST, то ничего не 1 будет изображено. Это признак того, что память пуста. **Команда NEW (а также SCRATCH или UNSAVE)** *Общая форма записи.* NEW
У этой команды могут быть и другие имена, например SCRATCH *или* I UNSAVE *Она освобождает память от операторов программы на Бейсике, но не уничтожает какую-либо часть системы с Бейсиком.*
2.4.4. КОМАНДА OLD (ИЛИ GET, ИЛИ LOAD)
Если эта команда имеется, то с ее помощью можно найти поименованную программу в архивной памяти и загрузить эту программу в память. Перед загрузкой поименованной программы эта команда всегда удаляет текущее содержание памяти. Проверьте ее действие, указав после команды OLD (или GET, или LOAD) команду LIST.

При работе с ЭВМ ICL 2904 команда
GET FRED
загрузит в память из файла на диске копию программы по имени FRED. При работе с микроЭВМ (CBM PET) команда
LOAD "FRED"

загрузит в память с кассетного магнитофона программу по имени FRED; обратите внимание на кавычки, требующиеся в этой версии. Ниже приводятся основные варианты этой команды (дополнительные детали см. в разд. 8.1). Подобно команде SAVE (см. подразд. 2.4.5), эта команда очень зависит от конкретной ЭВМ, так как существенным образом использует свойства ее аппаратуры и то, насколько легко управлять аппаратурой посредством обращений к операционной системе. Под аппаратурой здесь понимается широкий спектр архивной памяти: от дисководов для жестких магнитных дисков способных хранить 200 млн. байт, до кассетных магнитофонов и дисководов для гибких дисков.

Команда OLD (а также GET или LOAD)

Для больших ЭВМ, обычно с мультитерминалъной системой:

OLD *имя _ файла или*

GET *имя_файла*

Для микроЭВМ с дисками:

LOAD *"имя_файла" или*

LOAD *"имя_файла",*R

Для микроЭВМ, имеющих только кассетный магнитофон:

LOAD *или*

LOAD *"имя_файла "*

"Имя_файла" - имя файла, содержащего программу. По этой команде происходят освобождение памяти и загрузка в нее программы, ранее запомненной в файле с помощью команды SAVE

Если в команде дополнительно указан необязательный параметр R, *то те файлы данных, с которыми работала предыдущая программа, останутся открытыми и тем самым будут доступны новой программе, загружаемой и исполняемой по этой команде.*

Поскольку кассетный магнитофон представляет собой устройство с последовательным доступом к информации, то команда "LOAD без указания имени файла однозначно воспринимается как "загрузка с магнитной ленты следующей программы". В некоторых системах после имени файла может быть Указан номер устройства.

В Бейсике Microsoft есть команда MERGE (слияние), которая похожа на LOAD, но не освобождает память перед загрузкой программы. С ее помощью можно объединять программы или части программ. В этой версии Бейсика команда RUN дополнительно предоставляет возможность предварительной загрузки и исполнения программы. Для этого команда задается в виде

RUN "имя_файла",R

Здесь R — необязательный параметр, значение которого обсуждалось выше. 2.4.5. КОМАНДА SAVE

Эта команда — обратная по отношению к команде OLD в том смысле, **что** она копирует программу из памяти ЭВМ в архивную память. Как и в случае команды OLD, тип архивной памяти во многом определяет варианты команды (см. разд. 8.1).

В большинстве систем в команду SAVE включается имя файла, в котором должна быть запомнена программа. Однако, по крайней мере, у одной большой ЭВМ имеется команда NAME, позволяющая отдельно назвать имя той программы, которая затем должна быть сохранена по команде SAVE со специальными параметрами, позволяющими другим пользователям системы иметь к этой программе доступ.

2.4.5. Команда SAVE

Для больших ЭВМ: SAVE имя_файла *или*

SAVE дополнительные параметры (для сохранения программы, имя которой ранее указано командой NAME)

Для микроЭВМ с диском или кассетным магнитофоном:

SAVE "имя_файла"

Для микроЭВМ с кассетным магнитофоном:

SAVE

Эта команда копирует текущую программу из памяти ЭВМ в архивную память и сохраняет ее там в виде файла с заданным именем.
Если указаны "дополнительные параметры ", то тем самым определен тип доступа к Вашей программе, разрешаемого другим пользователям мулътитерминальной системы.

2.5. СОХРАНЕНИЕ ПРОГРАММЫ

При работе с вычислительной системой очень важно иметь возможность запоминать в архивной памяти находящиеся в памяти ЭВМ программы и данные и считывать их обратно. Архивной памятью служит магнитный носитель, способный хранить информацию многие месяцы и даже годы, не требуя для этого электропитания. Как и в случае машинных адресов памяти, сущест-

вуют довольно сложные средства выбора требуемых областей магнитного носителя. На магнитных лентах имеются дорожки, а на дисках — дорожки, секторы или даже цилиндры. К счастью, есть средства отнести все эти понятия к сфере компетенции ЭВМ и работать с архивной памятью гораздо более удобным образом. Большинство систем с Бейсиком позволяет представлять эту память как большую область, способную сохранять информацию. Любой части этой области можно присвоить имя по своему усмотрению (выбор имени ограничен лишь некоторыми соглашениями), и в поименованной области можно запоминать программы или данные. Каждая поименованная часть называется файлом, а вся область — файловой.

2.5.1. ПОЛУЧЕНИЕ ДЕТАЛЬНЫХ СВЕДЕНИЙ О ФАЙЛАХ

Объем памяти, требуемый для конкретного файла, зависит от объема содержащейся в нем программы (или данных), но обычно он не может быть меньше некоторого минимального объема, определяемого характеристиками системы. Так как архивная память не может содержать бесконечное число файлов, то Вам полезно знать, что она содержит и сколько именно памяти уже занято. Если в Вашем распоряжении имеется только кассетный магнитофон, то Вам придется хранить на бумаге список запомненных на кассете файлов с указанием их положения на магнитной ленте, но если Вы располагаете дисковой вычислительной системой, то в ней существует встроенный справочник. Операционная система или система с Бейсиком отражает в нем текущее состояние дисковой памяти.

По поводу команд, обеспечивающих выдачу сведений обо всех этих деталях, обратитесь к справочнику по Вашей ЭВМ. Некоторые из широко распространенных вычислительных систем освещаются в гл. 8.

2.6. ПРИМЕР СЕАНСА РАБОТЫ

В приведенном ниже примере сеанса работы с Бейсиком опущены детали процесса инициации системы, описанного в предыдущих разделах этой главы. Разрабатываемая здесь программа требует ввести число и выводит квадрат, куб и четвертую степень этого числа.

Вводимые пользователем данные подчеркнуты. В скобках в конце строки или между строк даются комментарии.

Инициация системы с Бейсиком

10 REM NUMBER EXAMPLE (набирается каждая строка)
20 INPUT A
30 B=A*A 10 PRINT "КВАДРАТ РАВЕН" ,A
LIST (печатается текущее состояние программы)


```
10  REM NUMBER EXAMPLE
20  INPUT A
30    B=A*A
40  PRINT "КВАДРАТ РАВЕН" ,A
```

```
RUN                                    (исполнить программу)
? 7                                    (запрашивается ввод)
КВАДРАТ РАВЕН   7                      (печатается результат)
ERROR AT LINE   40
```
(Системе могло "не понравиться", что программа не завершена оператором END. Она может не допускать выполнение команды RUN до тех пор, пока не будет вставлен оператор END. А результат неправилен потому, что в строке 40 вместо переменной B была указана переменная A.)
```
40   PRINT "КВАДРАТ РАВЕН", B          (исправление строки 40)
50     B=B*A                           (набираются новые строки)
60   PRINT "КУБ РАВЕН", B
70     B=B*B
80   PRINT "ЧЕТВЕРТАЯ СТЕПЕНЬ", B
90 PRINT
100   END
RUN                                    (исполнить программу)
? 9                                    • (запрашивается ввод)
КВАДРАТ РАВЕН 81
КУБ РАВЕН    729
ЧЕТВЕРТАЯ СТЕПЕНЬ   5.31441E+5       (печатаются результаты)
END AT LINE 100
```
(Правильными будут результаты 81, 729 и 6561, поэтому надо искать ошибку в программе.)
```
LIST                                   (распечатать программу)
10      REM NUMBER EXAMPLE
11      20   INPUT A
    30     B=A*A
    42
```

```
40   PRINT "КВАДРАТ РАВЕН", B
50     B=B*A
60   PRINT "КУБ РАВЕН", B
70     B=B*B
80   PRINT "ЧЕТВЕРТАЯ СТЕПЕНЬ", B
90   PRINT
100   END
```
(Ошибочна строка 70; в ней должно быть B*A. Кроме того, надо поправить текст в строке 80.)
```
70 B=B*A                               (строки набираются заново)
80   PRINT "ЧЕТВЕРТАЯ СТЕПЕНЬ РАВНА", B
RUN                                    (исполнить программу)
? 9                                    (запрашивается ввод)
КВАДРАТ РАВЕН    81
КУБ РАВЕН          729                 (правильные результаты)
ЧЕТВЕРТАЯ СТЕПЕНЬ РАВНА 6561
```
(пустая строка выдана оператором PRINT в строке 90) END AT LINE 100
```
SAVE "EXAMPLE 1"                       (сохранить копию текущей
                                       программы в архивной памяти
          под именем EXAMPLE 1)
LIST 20                                (распечатать одну строку про-
                                       граммы)
20  INPUT A                            (программа все еще находится
```
в памяти; команда SAVE не перемещает ее, а только копирует)
```
NEW                                    (удалить программу из памяти)
```

LIST
NO PROGRAM (так отвечают некоторые системы)
(Если в памяти нет никакой программы, реакции на команду LIST может и не быть.)
LOAD "EXAMPLE 1" (восстановить копию ранее сохраненной программы)
43

LIST 80,100 (распечатать группу строк программы)
80 PRINT "ЧЕТВЕРТАЯ СТЕПЕНЬ РАВНА" ,B 90 PRINT 100 END
Программа возвращена в память и может быть исполнена снова, а может быть и изменена с какой-либо целью. Учтите, что некоторые детали команд SAVE и LOAD зависят от используемого носителя (ленты или диска), а также от вычислительной системы.
УПРАЖНЕНИЯ
2.1. Выплата денег по ссуде или закладной производится фиксированными ежегодными платежами, погашающими как часть полученной взаймы суммы, так и начисленные за непогашенный остаток проценты.
Если *m* фунтов стерлингов взято взаймы на t лет с процентами i, то размер годового платежа можно определить по следующей алгебраической формуле:
размер платежа=$mr^t(r-1)/(r^t-1)$, где $r=1+i/100$.
Напишите программу, требующую ввода значений *m, t, i* и вычисляющую размер годового платежа.
2.2. Требуется производить картонные коробки с заданными размерами. Определите объем коробки, площадь поверхности расходуемого на нее картона и стоимость производства.
Длина коробки *L,* высота *H* и ширина *W* (в сантиметрах) вводятся в программу. За раскрой картона и формирование коробки из заготовки взимается постоянная плата 5 пенсов, не зависящая от размеров коробки. Цена самого картона 0.02 пенса за квадратный сантиметр. Припуском картона на склейку стенок пренебрегите. Напишите программу так, чтобы печаталось все, что должно вводиться и выводиться.
2.3. Пусть коробки из упражнения 2.2 надо использовать для упаковки шоколадных конфет, для чего на крышке и четырех боковых стенках каждой коробки требуется сделать цветную надпечатку. Стоимость надпечатки 0.005 пенса за квадратный сантиметр.
Модифицируйте программу из упражнения 2.2 с учетом этих изменений в условии задачи. Выберите размеры коробки с таким расчетом, чтобы в ней помещался 1 фунт (около 450 г) конфет, и проанализируйте, как изменяется стоимость производства коробки, если ее размеры варьируются так, чтобы объем коробки оставался одним и
тем же.
2.4. Требуется ежемесячно проводить расчеты с владельцами кредитных карточек. Общая форма расчета показана ниже. Она начинается с баланса предыдущего месяца (т. е. непогашенного владельцем карточки долга) и начисленных на него процентов (из расчета 8 % ежемесячно). Далее идет перечень купленных владельцем карточки товаров с указанием их стоимости, В конце подытоживается текущая сумма долга клиента кредитной фирме и указывается минимальная выплата для погашения долга, составляющая 10 % от него.
Напишите программу для выдачи подобного расчета после покупки трех товаров. (Для хранения названий товаров используйте строковые переменные.)
44
Расчет должен выдаваться в следующем виде: КРЕДИТНАЯ ФИРМА TAXHAVEN

ПРЕДЫДУЩИЙ БАЛАНС:	XXXX
ПРОЦЕНТЫ:	XXX
ТОВАР	СТОИМОСТЬ

YYYYYY	XXXX
YYYYYY	XXXX
YYYYYY	XXXX
ИТОГО:	XXXX

МИНИМАЛЬНАЯ ВЫПЛАТА	:	XXXX

3. ДРУГИЕ ОСНОВЫ ПРОГРАММИРОВАНИЯ НА БЕЙСИКЕ

В первых двух главах были представлены элементы языка Бейсик и описаны приемы работы на ЭВМ. В рассматривавшихся до сих пор программах исполнение начиналось с оператора, имеющего наименьший номер, а заканчивалось оператором с наибольшим номером. До сих пор ничего не говорилось о средствах пропуска оператора при выполнении некоторых условий или передачи управления на начало программы после достижения ее конца. Подобные изменения хода исполнения программы можно сделать с помощью четырех операторов Бейсика, а именно IF, GO TO и FOR-NEXT. В настоящей главе описаны общие для всех систем с Бейсиком формы этих операторов.

Так как действие этих операторов выражается в передаче управления другой части программы, важно иметь какие-либо средства иллюстраций таких Действий. Одно из старых, но не очень удобных средств - так называемые блок-схемы. В следующем разделе представлено лучшее средство — структограммы.

3.1. СИМВОЛЫ СХЕМЫ УПРАВЛЕНИЯ ПРОГРАММОЙ

Приводимые ниже символы достойны широкого распространения в качестве лучшего средства графического изображения процесса передачи управления (исполнения) в структурированной программе. Будучи подходящим разом скомбинированы, они дают полное визуальное представление деталей логических операций над группами операторов программы. Такие пред-

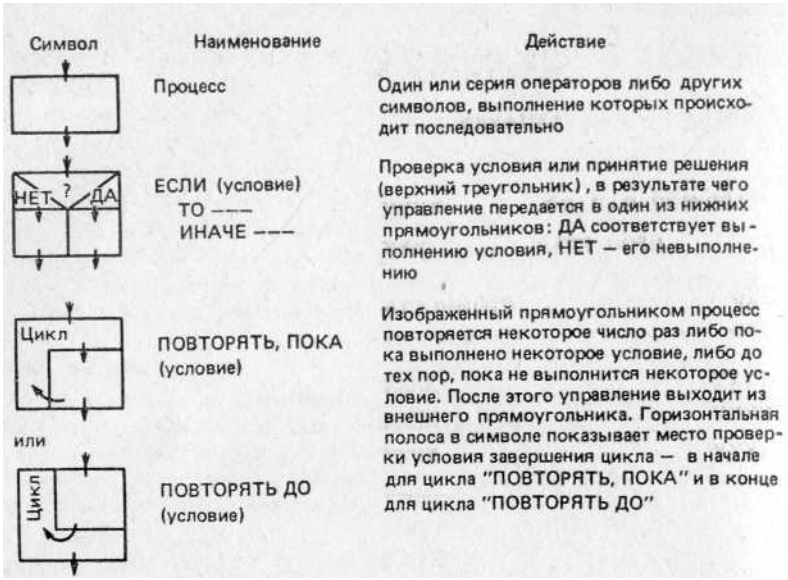


Рис. 3.1. Некоторые символы структограмм. Стрелки не являются частью символов; они добавлены здесь для того, чтобы помочь начальному усвоению значения символов

ставления называются структограммами или, по имени их авторов, диаграммами Нэсси—Шнейдермана.

Сами по себе они не являются методом разработки программ, но служат неплохим подспорьем при разработке. Они представляют хорошую систему описания и понимания программ и используются в этой главе для иллюстрации процесса передачи управления в программе.

Простейшим символом является прямоугольник, содержащий описание действий, выполняемых оператором Бейсика или группой таких операторов. Такой прямоугольник называется *символом процесса.* В подобном символе можно помещать операторы присваивания, ввода-вывода, вызова или любой другой набор последовательно выполняемых операторов. Как показано на рис. 3.1, управление передается в прямоугольник сверху, а выходит из него снизу.

Хотя это не представит значение символов в выгодном свете, рассмотрим простую программу из разд. 2.1. Ее действия можно изобразить символами процесса следующим образом:

Ввод значений A и B
Вычисление суммы
Изображение результатов
Конец

В этой простой программе, состоящей исключительно из последовательно исполняемых операторов, разделение действий по различным прямоугольникам можно производить как угодно, поэтому в отдельные прямоугольники были выделены логически завершенные шаги программы: ввод, присваивание вывод и завершение работы. В принципе все эти действия можно было бы включить в один символ процесса. Следует обратить внимание на то обстоятельство, что полная программа сама представляет собой блок — большой символ процесса, содержащий внутри себя другие символы. Это характерно для данного способа описания программы и сродни структурному программированию.

Управление начинает свой путь на верхней стороне внешнего прямоугольника, проходит через каждый прямоугольник и завершает путь на нижней стороне внешнего прямоугольника. Схема законченной программы или ее логически завершенной части должна умещаться на одной странице. В отличие от блок-схем, здесь нет средств ссылки на другую страницу, поскольку они могли бы вызвать недоразумение.

3.2. ПРОСТЫЕ ОПЕРАТОРЫ IF

Обсуждавшиеся до сих пор программы имели очень простую структуру — последовательность действий, подобную иллюстрированной в приведенном выше примере в разд. 3.1. Эта структура, или, скорее, отсутствие содержательной структуры, не позволяет решить многие задачи. Однако в дополнение к ней существует возможность производить проверку выполнения условий с помощью оператора IF, позволяющую разрабатывать более разносторонние программы.

Рассмотрим следующую программу:

```
10 INPUT A, B
20 IF A>B THEN 50
30   PRINT A,B
40 STOP
50   PRINT B,A
60 END
```

Ее действия таковы: вводятся два числа и печатаются в порядке возрастания. В строке 20 проверяется условие "A больше B"; если оно выполнено, то управление передается строке 50, в которой меньшее из двух значений (B) печатается первым. Если это условие не выполнено, то управление передается следующему оператору (строке 30), который первым печатает значение A. Обратите внимание на оператор **STOP,** завершающий исполнение программы.

Оператор **STOP**

Общая форма записи:

STOP

Вызывает завершение исполнения программы; может указываться многократно или вообще не присутствовать. Обычно этим оператором нельзя заменить оператор END, *являющийся последним оператором программы.*

Когда при исполнении программы очередь доходит до оператора STOP, то система прекращает исполнение программы и выдает приглашение к вводу другой команды. Обычно в программе должен быть только один оператор END (при этом он должен иметь наибольший номер), но в некоторых системах разрешается многократное использование оператора END в качестве
эквивалента оператора STOP.

Таким образом, оператор IF позволяет осуществить условное разветвление управления, т. е. его передачу другой части программы в случае выполнения некоторого условия.
Ниже приводится программа, использующая две из возможных конструкций условия в операторе IF. Эта программа должна печатать сообщение, указывающее знак введенного числа.

```
10 PRINT "ВВЕДИТЕ ЧИСЛО"
20 INPUT N
30 IF N≥0 THEN 60
40   PRINT "ВАШЕ ЧИСЛО ОТРИЦАТЕЛЬНО"
50 STOP
60 IF N<>0 THEN 90
70   PRINT "ВАШЕ ЧИСЛО - НУЛЬ"
80 STOP
90   PRINT "ВАШЕ ЧИСЛО ПОЛОЖИТЕЛЬНО" 100 END
```

Простой оператор IF

Общая форма записи:

IF e THEN s

Здесь e - *условие, которое может быть выполнено или нет, а s- номер оператора.*

Если условие e *истинно, то управление передается оператору с номером* s. *В противном случае исполняется оператор, номер которого следующий за номером оператора* IF.

В некоторых системах дополнительно разрешается форма

IF e GO TO s *имеющая ровно тот же смысл, что и приведенная выше.*

В других системах, например в Бейсике BBC, после THEN *разрешается указывать любой оператор Бейсика (кроме* IF). *Например:*

```
IF A=2+B THEN PRINT    "СУММА РАВНА" ,A
IF X>Y THEN STOP
```

Если Ваша система с Бейсиком позволяет указывать в одной строке несколько операторов, то будьте осторожны при применении оператора IF: если условие не выполнено, управление будет передано следующей строке, а не следующему оператору.

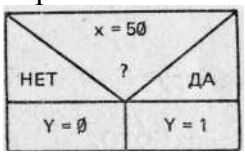
Ниже приведены допустимые формы оператора IF. Если условие выполняется, то управление передается оператору с номером s:

Форма		*Условие*
IF a>b	THEN s	a больше b
IF a<b	THEN s	a меньше b
IF a=b	THEN s	a равно b
IF a <>b	THEN s	a не равно b
IF a≥b	THEN s	a больше b или
IF a =>b	THEN s	равно ему
IF a≤b	THEN s	a меньше b или
IF a=<b	THEN s	равно ему

Зачастую оператор IF используется для присваивания переменной одного из двух значений. Например, приведенные ниже операторы программы присваивают переменной Y значение 1, если X = 50, и значение 0 в противном случае (в строке 70 оператор GO TO передает управление строке 100; подробнее об этом операторе см. подразд. 3.2.1) .

```
50  IF X = 50 THEN 80
60     Y=0
70  GO TO 100
80     Y=1
100----
```

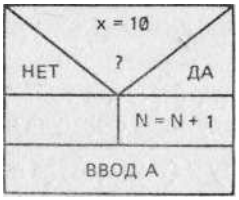
Обращаясь к рис. 3.1, приведенные выше операторы можно проиллюстрировать следующим образом:



Если условие (X = 50) выполнено, то управление передается в правый нижний прямоугольник; в противном случае —в левый нижний прямоугольник. Происходящие в строке 100 действия должны быть изображены под этими прямоугольниками.

Если результатом выполнения оператора IF может быть только одно действие, то один из нижних прямоугольников надо оставить пустыми. Например, операторы

40 IF X=10 THEN N=N+1 60 INPUT A можно проиллюстрировать следующей структограммой:



Условие в операторе IF может содержать довольно сложные выражения, например:

IF A+B>43 THEN 210

IF A/C≤B+100+D THEN 92

IF 10=A+B THEN A=B/4

Условные выражения можно комбинировать с помощью логических операций NOT (НЕ), AND (И), OR (ИЛИ). Обычно смысл этих операций нетрудно определять, читая выражение, но при необходимости иметь более полные сведения см. подразд. 6.1.1. Приведем пример применения этих операций:

IF X>1 AND Y>1 THEN 90 IF NOT (X>2) THEN STOP IF X>1 OR X<0 THEN 100

3.2.1. ОПЕРАТОР GO TO

Оператор GO TO всегда вызывает передачу управления указанному в нем оператору.

Оператор GO TO

Общая форма записи:

GO TO s *или*

GO TO s *(запись одним словом)*

Управление передается оператору с номером s.

Учтите, что в Бейсике BBC не допускается первая форма записи (с пробелом) и разрешается только форма GO TO.

Приведем примеры этого оператора, позволяющего "перескакивать" назад или вперед:

GO TO 50

GO TO 9999

Обратившись к рис. 3.1, Вы можете заметить, что для оператора GO TO не предусмотрено никакой диаграммы. Причина этого проста: оператор не принадлежит к специальным образом подобранному набору структур, обеспечивающему разработку программы любого типа. Используйте оператор GO TO с большой осторожностью. Лучше всего сочетать его с другими операторами так, чтобы в результате образовывались стандартные структуры, изображенные на рис. 3.1.

3.2.2. ПРИМЕР ПРОГРАММЫ

Рассмотрим программу для перевода температуры из шкалы Цельсия в шкалу Фаренгейта. Соответствующая формула имеет вид

Температура по Фаренгейту = (температура по Цельсию)*180/100 + 32.

Можно написать такую программу:

```
10   INPUT C
20     F=C*180/100+32
30        PRINT  C, F 40  END
```

Она выдает вполне удовлетворительный результат, но от Вас требуется помнить, что символизируют C и F и в каком порядке они печатаются. Было бы лучше печатать пояснения, для чего можно добавить строки:

```
30 PRINT   "ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ =";C
35 PRINT   "ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ =";F
6 PRINT    "ПРОГРАММА ПРЕОБРАЗОВАНИЯ ТЕМПЕРАТУРЫ ИЗ ГРАДУСОВ ЦЕЛЬСИЯ"• $
PRINT   " В ГРАДУСЫ ФАРЕНГЕЙТА"                    ГРАДУСОВ ЦЕЛЬСИЯ ,
7 PRINT   "УКАЖИТЕ ТЕМПЕРАТУРУ В ГРАДУСАХ ЦЕЛЬСИЯ"
```

Измененная программа выводит результаты в следующем виде (ответы пользователя подчеркнуты):

```
RUN
ПРОГРАММА ПРЕОБРАЗОВАНИЯ ТЕМПЕРАТУРЫ ИЗ ГРАДУСОВ ЦЕЛЬСИЯ В ГРАДУСЫ ФАРЕНГЕЙТА
УКАЖИТЕ ТЕМПЕРАТУРУ В ГРАДУСАХ ЦЕЛЬСИЯ
ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ =50 ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ=122
END AT LINE 40
```

Будучи полезной для преобразования одного значения температуры, эта программа не очень удобна для преобразования ряда значений, так как ее придется запускать столько раз, сколько всего значений. Чтобы программа могла обработать последовательность значений, надо после исполнения оператора печати в строке 35 вернуть управление на строку 6. Добавив строку

```
37  GOTO 6
```

и три оператора REM, получим

```
LIST
1 REM ПРЕОБРАЗОВАНИЕ ТЕМПЕРАТУРЫ ИЗ ГРАДУСОВ ЦЕЛЬСИЯ
2 REM В ГРАДУСЫ ФАРЕНГЕЙТА
3 REM C=ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ, F=ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ
5 PRINT "ПРОГРАММА ПРЕОБРАЗОВАНИЯ ТЕМПЕРАТУРЫ ИЗ ГРАДУСОВ ЦЕЛЬСИЯ"
6 PRINT " В ГРАДУСЫ ФАРЕНГЕЙТА"
7 PRINT "УКАЖИТЕ ТЕМПЕРАТУРУ В ГРАДУСАХ ЦЕЛЬСИЯ"
10   INPUT  C
20     F=C*180/100+32
30 PRINT "ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ =";C
35 PRINT "ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ =";F
37 GO TO 6
40 END
51
```

```
ПРОГРАММА ПРЕОБРАЗОВАНИЯ ТЕМПЕРАТУРЫ ИЗ ГРАДУСОВ ЦЕЛЬСИЯ
В ГРАДУСЫ ФАРЕНГЕЙТА
УКАЖИТЕ ТЕМПЕРАТУРУ В ГРАДУСАХ ЦЕЛЬСИЯ
?50
ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ=50
ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ=122
УКАЖИТЕ ТЕМПЕРАТУРУ В ГРАДУСАХ ЦЕЛЬСИЯ
ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ =80 ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ =176
УКАЖИТЕ И Т. Д.
```

Эта программа будет без конца повторяться между строками 6 и 37. Воспользовавшись конструкциями рис. 3.1, получим для нее диаграмму следующего вида:

ПЕЧАТЬ заголовка Повторять без конца
ПЕЧАТЬ запроса на ввод
ВВОД С
F=C*180/100+32
ПЕЧАТЬ результатов
КОНЕЦ

Остановиться можно, только прервав исполнение программы. Обычно для этого имеется специальная команда. Однако такой ситуации нельзя допускать при разработке хорошей программы; поэтому сделаем так, чтобы программа завершала свою работу при получении определенного значения С, скажем 9999. Добавив оператор

```
15   IF C =9999 THEN STOP получим
LIST
1  REM ПРЕОБРАЗОВАНИЕ ТЕМПЕРАТУРЫ ИЗ ГРАДУСОВ ЦЕЛЬСИЯ
2 REM В ГРАДУСЫ ФАРЕНГЕЙТА
3 REM C=ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ. F=ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ
6 PRINT "ПРОГРАММА ПРЕОБРАЗОВАНИЯ ТЕМПЕРАТУРЫ ИЗ ГРАДУСОВ ЦЕЛЬСИЯ";
6 PRINT " В ГРАДУСЫ ФАРЕНГЕЙТА"
7 PRINT "УКАЖИТЕ ТЕМПЕРАТУРУ В ГРАДУСАХ ЦЕЛЬСИЯ" 10 INPUT C
15    IF C=9999 THEN STOP
2B   F=C*180/100+32
38 PRINT "ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ=";C
35 PRINT "ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ=";F
37 GOTO 6
40 END
RUN
ПРОГРАММА ПРЕОБРАЗОВАНИЯ ТЕМПЕРАТУРЫ ИЗ ГРАДУСОВ ЦЕЛЬСИЯ В ГРАДУСЫ ФАРЕНГЕЙТА
УКАЖИТЕ ТЕМПЕРАТУРУ В ГРАДУСАХ ЦЕЛЬСИЯ
ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ - 50 ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ=122
52



УКАЖИТЕ ТЕМПЕРАТУРУ В ГРАДУСАХ ЦЕЛЬСИЯ
?80
ТЕМПЕРАТУРА  ПО  ЦЕЛЬСИЮ  =  80
ТЕМПЕРАТУРА  ПО  ФАРЕНГЕЙТУ  =176
УКАЖИТЕ  ТЕМПЕРАТУРУ  В  ГРАДУСАХ  ЦЕЛЬСИЯ
?9999
END  AT  LINE  40
```

Выполняемые этой программой действия можно изобразить следующей диаграммой:

Эта диаграмма гораздо лучше предыдущей, так как обладает точкой завершения работы.

3.3. ЦИКЛЫ

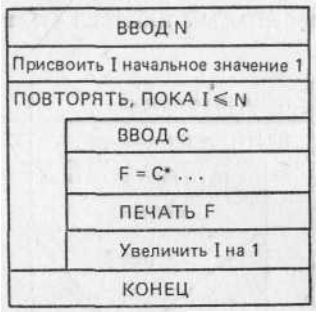
В предыдущем разделе было показано, как можно обеспечить повторение исполнения последовательности операторов и управлять завершением полученного таким образом цикла.

Если нам заранее известно, сколько значений температуры требуется преобразовать, то в процессе преобразования можно подсчитать число преобразованных значений и перейти на конец программы, как только будет выполнено заданное число преобразований. Такой цикл можно организовать с помощью оператора IF, переменной I (счетчика цикла), регистрирующей число выполненных повторений, и переменной N, содержащей требуемое число повторений:

```
10 INPUT N 20 I=1
30 IF I>N THEN 90
40   INPUT C
50   F=C*180/100+32
60   PRINT F
70   I=I+1
80 GOTO 30
90 END
```

Не считая ввода значения N, для управления циклом используются следующие операторы: строка 20, которая устанавливает счетчик I равным 1, строка

30, которая проверяет, не сделано ли уже N повторений, строка 70, которая при каждом проходе цикла увеличивает счетчик I на 1, а также строка 80, которая возвращает управление на начало цикла - строку 30. И вновь оператор GO TO надо представлять в структограмме только как элемент цикла, так что эту программу можно изобразить в следующем виде:



В этой диаграмме смущает потеря оператора IF. Она произошла потому, что этот оператор вошел в конструкцию "ПОВТОРЯТЬ, ПОКА". Горизонтальная черта показывает, что проверка условия происходит перед выполнением оператора INPUT C (ВВОД С) .

Кроме указанного метода организации цикла есть еще несколько способов. Например, удалите строки 30 и 80 и вставьте строку

```
80 IF I≤N THEN 40
```

которая проверяет счетчик цикла в конце каждого вычисления значения температуры по Фаренгейту. Структограмма пересмотренной программы примет следующий вид:

ВВОД N
Присвоить I начальное значение 1
ВВОД С F=C* ... ПЕЧАТЬ F Увеличить I на 1
ПОВТОРЯТЬ ДО I> N
КОНЕЦ

И вновь оператор IF, на этот раз замыкающий цикл, поглощен конструкцией "ПОВТОРЯТЬ ДО". Многие языки программирования, в том числе некоторые расширенные версии Бейсика, содержат особые операторы для облегчения выполнения подобных циклов, что позволяет избежать несколько "неуклюжего" употребления операторов IF и GOTO.
Первая из рассмотренных форм организации циклов соответствует действию операторов FOR-NEXT, описанных в следующем разделе.

3.3.1. ОПЕРАТОРЫ FOR-NEXT

В языке Бейсик предусмотрена пара операторов, обеспечивающих построение цикла того же вида, что и в приведенном выше примере (см. разд. 3.3). Это операторы FOR и NEXT, которые окаймляют группу операторов, составляющих тело цикла. Например, приведенную выше программу можно с их помощью переписать в виде

```
10 INPUT N
20 FOR I=1 TO N    -----!
30   INPUT C            !
40   F=C*180/100+32     !      цикл
50   PRINT F            !
60   NEXT I        -----!
90 END
```

Строка 20 автоматически берет на себя присваивание счетчику цикла I начального значения 1 и увеличение I на 1 после каждого прохода цикла. Строка 60 обозначает конец цикла.
Все операторы между операторами FOR и NEXT исполняются при I=1, I = 2,I = 3 и так далее до I = N. После этого управление передается оператору, следующему за оператором NEXT. Следовательно, в нашем примере будет введено, преобразовано и напечатано N значений температуры, после чего программа завершает свою работу.
Эту программу можно иллюстрировать следующей диаграммой:

ВВОД N
ДЛЯ I=1 ДО N
ВВОД С
F=C* ...
ПЕЧАТЬ F
КОНЕЦ

Оператор FOR
Общая форма записи: FOR *i=j* TO *k или*
FOR *i =j* TO *k* STEP *m*
где i - числовая управляющая переменная, а j, k, m могут быть числовыми выражениями (т. е. константами, переменными, а также комбинациями переменных и констант).
Нижней границей цикла, начатого оператором FOR, *служит оператор* NEXT i. *Все операторы между* FOR *и* NEXT *выполняются при*
i=j, i=j+m, i =j+2m, i =j+3m, i =. . . до тех пор, пока не будет достигнут предел k.

Другими словами, если m положительно, то цикл продолжается все время пока i≤k. Если m отрицательно, то цикл продолжается, пока i≥k. Если дополнительное служебное слово STEP *(шаг) отсутствует, то величина шага m предполагается равной +1.*

Приведем несколько примеров. Программа

```
10   FOR I = 1 TO 10
20     PRINT I
30   NEXT I
```

напечатает все числа от 1 до 10, Программа

```
10  FOR K=1 TO 10 STEP 2
20     PRINT K
30  NEXT K
```

напечатает все нечетные числа от 1 до 10. Программа

```
10   X=2
20  Y=3
30   FOR P = X+Y TO X*Y
40     PRINT P
50  NEXT P
```

напечатает значения 5 и 6, а программа

```
10   FOR A = 10 TO  1  STEP -1
20     PRINT A
30  NEXT A
```

напечатает числа от 10 до 1 в порядке убывания.

Во многих системах с Бейсиком, например Sinclair ZX81, проверка выхода за пределы цикла осуществляется согласно стандарту, в начале цикла. Поэтому при исполнении операторов

```
10  FOR I=3 TO 2
20     PRINT I
30  NEXT I
```

ничего не будет напечатано и управление будет передано в строку 40. Это соответствует примеру программы из разд. 3.3 и структограмме "ПОВТОРЯТЬ, ПОКА".

В других системах с Бейсиком, например BBC, стандарт не соблюдается. Цикл выполняется, по крайней мере, один раз, и проверка его завершения производится в конце цикла. Подобному подходу отвечает модифицированная программа из разд. 3.3 и структограмма "ПОВТОРЯТЬ ДО". В этом случае приведенная выше программа напечатает 3.

Во всех приводимых ниже примерах предполагается, что цикл FOR-NEXT реализован согласно структограмме "ПОВТОРЯТЬ, ПОКА", поскольку именно этот вариант предусмотрен стандартом для минимального подмножества языка Бейсик. Если в Вашей системе проверка условия выхода из

цикла производится в конце цикла, то надо пользоваться другой структограммой, "ПОВТОРЯТЬ ДО", помечая ее внизу служебным словом FOR. Можете не очень заботиться об этом различии: оно вызывает изменения в результатах работы программы при переходе от системы первого типа к системе второго типа примерно в одном случае из тысячи. Достаточно лишь иметь представление о том, к какому из этих двух типов относится Ваша система. Дадим несколько предостережений. Не изменяйте значение управляющей переменной внутри цикла. Система автоматически увеличивает значение этой переменной и проверяет его при каждом проходе цикла. Поэтому управляющая переменная не должна появляться в левой части оператора LET, в операторах READ или INPUT, а также не должна использоваться как управляющая переменная вложенного цикла. Ниже приводятся примеры ошибочного использования управляющей переменной:

```
10  FOR I=1 TO 10
20     I=I+1                    (управляющая переменная I изменяется)
30 NEXT I
10  FOR A = 2 TO 8 STEP 2
20 INPUT I,A      (управляющая переменная указана в операторе INPUT)
30     I=10+I+A
40 NEXT A
```

Не следует рассчитывать на то, что после выхода из цикла управляющая переменная будет иметь определенное значение. В примере

```
10 FOR L=1 TO 10
20     PRINT L
30 NEXT L
40 PRINT L
```

при исполнении строки 40 может печататься 10,11 или вообще какая-нибудь ерунда. Результаты будут отличаться у разных систем, хотя в стандарте минимального подмножества Бейсика рекомендуется, чтобы управляющая переменная после выхода из цикла содержала значение, которое она имела бы внутри цикла при его продолжении еще на один шаг.

Старайтесь внутри цикла FOR не изменять указанные в операторе FOR пределы значений управляющей переменной, хотя это может не повлиять на исполнение цикла; например, программа

```
10 A=2
20  FOR I = A TO 4
30     A=A+1
40     PRINT I
50 NEXT I
```

выдает правильные результаты 2, 3 и 4.

Оператор NEXT

Общая форма записи:

NEXT *i*

или

NEXT *где i — числовая переменная.*

Завершает цикл FOR, *Большинство версий Бейсика для каждого оператора* FOR *требует наличия отдельного оператора* NEXT *с указанием имени управляющей переменной. Однако в некоторых версиях допускается применение оператора* NEXT *без указания имени управляющей переменной или даже (в случае вложенных циклов) с указанием нескольких имен управляющих переменных.*

Для выхода из цикла FOR-NEXT, не дожидаясь его полного завершения, можно использовать операторы IF или GO TO. Правда, делать это не рекомендуется, так как такой выход из цикла может вызывать недоразумения и, кроме того, его, как правило, нельзя изобразить структограммой. И уж во всяком случае, никогда не передавайте управление внутрь цикла FOR-NEXT, так как это может привести к непредсказуемым результатам.

3.3.2. ВЛОЖЕННЫЕ ЦИКЛЫ FOR

В приведенном на рис. 3.2 примере показано, каким образом один цикл **FOR** можно использовать внутри другого. Обратите внимание на то, что внутренний цикл полностью содержится во внешнем. Такая структура называется вложением циклов FOR.

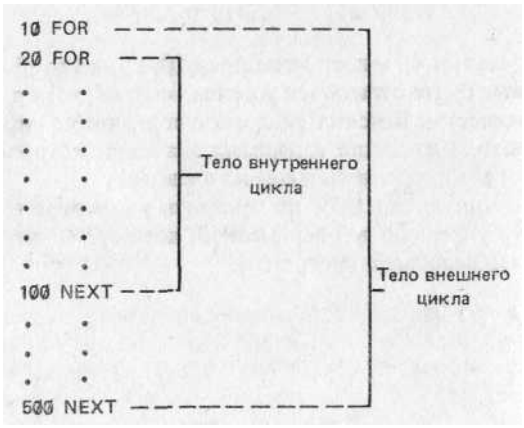
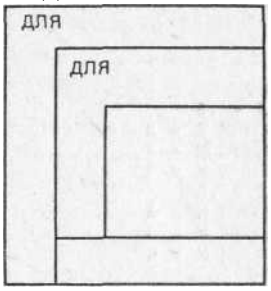


Рис. 3.2. Пример использования одного цикла FOR внутри другого

Подобное вложение изображается следующей диаграммой-



Циклы должны быть полностью вложены, так что внутренний цикл должен завершаться внутри внешнего. В следующей программе вложенные циклы используются для печати таблицы умножения:

```
10 FOR I=1 TO 5
20   FOR J=1 TO 5
30     Z=I*J
40     PRINT Z;
50   NEXT J
60   PRINT
70 NEXT I
```

Так как в операторе PRINT (строка 40) указана точка с запятой, то значения z будут печататься на одной строке до тех пор, пока управление не выйдет из внутреннего цикла, после чего оператор PRINT в строке 60 начнет новую строку для цикла, управляемого переменной I. Таким образом, вывод программы будет иметь следующий вид:

```
 1  2  3  4  5
 2  4  6  8 10
 3  6  9 12 15
 4  8 12 16 20
 5 10 15 20 25
```

Учтите, что в Бейсике BBC способ изображения числовых данных таков, что в приведенном выше примере столбцы результатов сольются, поэтому при работе с этой версией Бейсика строку 40 надо заменить на оператор

40 PRINT " ";Z;

Ниже демонстрируются различные возможности применения операторов F0R-NEXT на примере трех вложенных циклов:

```
20  B=4
30  FOR     I = 100  TO   200   STEP   60
40      PRINT   "ТЕКУЩЕЕ  ЗНАЧЕНИЕ   I=";I.
50   FOR J=.3 TO .2 STEP -.1
60    PRINT "ТЕКУЩЕЕ ЗНАЧЕНИЕ J=";J
70    FOR K=A TO B
80      PRINT "                         ТЕКУЩЕЕ ЗНАЧЕНИЕ К =";К
```

```
 90     NEXT K
100   NEXT J
110 NEXT I
120 END
RUN
ТЕКУЩЕЕ ЗНАЧЕНИЕ I = 100
     ТЕКУЩЕЕ ЗНАЧЕНИЕ J = .3
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 2
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 3
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 4
     ТЕКУЩЕЕ ЗНАЧЕНИЕ J = .2
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 2
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 3
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 4
ТЕКУЩЕЕ ЗНАЧЕНИЕ I = 160
     ТЕКУЩЕЕ ЗНАЧЕНИЕ J = .3
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 2
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 3
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 4
     ТЕКУЩЕЕ ЗНАЧЕНИЕ J = .2
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 2
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 3
          ТЕКУЩЕЕ ЗНАЧЕНИЕ K = 4
END AT LINE 120
```

Если возможно применение дополнительных форм оператора NEXT, то оператор NEXT без имени переменной отмечает конец цикла, начатого непосредственно предшествующим ему оператором FOR. В этом случае строки 90, 100, 110 можно переписать в виде

```
90  NEXT
100  NEXT
11.0  NEXT
```

А если допускается и вторая дополнительная форма, то эти строки можно заменить строкой

```
90  NEXT I, J, K
```

показывающей, что все циклы заканчиваются в одном и том же месте (первые переменные отвечают внешним циклам).

И, наконец, последнее: если используемые в операторе FOR переменные не являются целыми, то при изменениях значения управляющей переменной вычисления будут проводиться приближенно и момент завершения цикла может зависеть от точности вычислений, обеспечиваемой Вашей системой. Это очень неприятный момент, так как из-за этого могут время от времени получаться удивительные результаты (например, если в цикле с большим числом повторений используется шаг цикла с дробной частью).

3.3.3. ПРИМЕР ПРОГРАММЫ

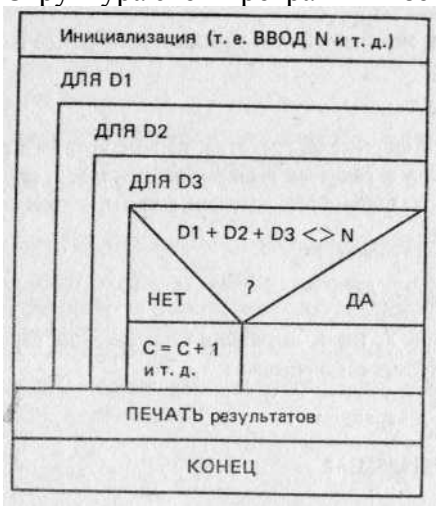
В этом подразделе иллюстрируется применение трех вложенных циклов. Приведенная здесь программа вычисляет вероятность получения заданной суммы очков при бросании трех игральных костей. Эта вероятность определяется как отношение числа тех вариантов, при которых выпадает заданная

сумма очков N, к числу всех вариантов (6*6*6 =216 вариантов). Варианты с заданной суммой очков подсчитываются в строке 80, а составляющие эту сумму очки печатаются в строке 90. Если сумма очков отличается от N, то происходит переход со строки 70 к строке 100, т. е. указанные выше действия обходятся. Ответы пользователя подчеркнуты:

```
10 INPUT N
20 PRINT
30 C=0
40 FOR D1=1 TO 6
50   FOR D2=l TO 6
60     FOR D3=l TO 6
70       IF(D1+D2+D3<>N) THEN 100
80       C=C+1
```

```
90      PRINT D1 ;D2 ;D3
100     NEXT D3
110   NEXT D2
120 NEXT D1
130 PRINT N;"ВЫПАДАЕТ В ";C;" СЛУЧАЯХ"
140 PRINT "ОТСЮДА ВЕРОЯТНОСТЬ";N; "РАВНА"; C/216
150 END
RUN
?6
1 1 4
1 2 3
1 3 2
1  1 2 1 3 2 2 2
2 3 1
3 1 2
3 2 1
4 1 1
6 ВЫПАДАЕТ В 10 СЛУЧАЯХ" ОТСЮДА ВЕРОЯТНОСТЬ 6 РАВНА 4.62963E-2 END AT LINE 150
```

Структура этой программы изображается следующей диаграммой:



При работе с Бейсиком ВВС строку 90 надо модифицировать следующим образом:

```
90 PRINT D1;" ";D2;" ";D3
```

3.4. СТАНДАРТНЫЕ ЧИСЛОВЫЕ ФУНКЦИИ

Во многих вычислениях используются действия, которые не могут быть простым способом сведены к арифметическим операциям +, -, *, /, ↑. Частным примером может служить извлечение квадратного корня из числа. В языке Бейсик предусмотрены специальные функции для выполнения некоторых таких действий. Например, оператор

```
20 A = SQR (X)
```

вычисляет квадратный корень из значения X и запоминает полученный результат в A. Функции типа SQR могут входить в состав выражений. При вычислении таких выражений имя функции заменяется на результат ее вычислений. Аргумент X приведенной выше функции можно заменить на выражение при условии, что результат его вычисления не будет отрицательным. Таким образом, допустимы следующие операторы:

```
20  A = SQR (A + B * 3.1) 20 A=B-SQR (14.7) 20  A=SQR(3+SQR(X))
```

В табл. 3.1 приводится рекомендованный перечень стандартных функций. Эти функции должны иметься во всех версиях Бейсика. Через X в этой таблице обозначается числовое выражение.

Таблица 3.1. *Стандартные числовые функции Бейсика*

Функция	Действие
ABS(X)	Дает абсолютное значение X, т. е. значение, получаемое после замены знака числа X на плюс: ABS(-3.7) = 3.7
ATN(X)	Дает значение арктангенса X, т. е. угла, тангенс которого равен X. Результатом является угол в радианах в диапазоне -π/2 ... π/2. Для перевода из радианов в градусы пользуйтесь соотношением 180 градусы=(180/π)* радианы где 180/π =57.29577951
COS(X)	Дает косинус угла X, где X задается в радианах. Для перевода из градусов в радианы пользуйтесь соотношением радианы=(π/180)*градусы где π/180=1.745329252E-2

Таблица 3.1 (окончание)

Функция	Действие
EXP(X)	Дает e (=2.718281828) в степени X, т. е. e^X. Обратите внимание на то, что e служит основанием натуральных логарифмов, а функции EXP и LOG являются взаимно обратными. Таким образом, LOG(EXP(X)) = X=EXP(LOG(X))
INT(X)	Дает наибольшее целое число, не превышающее X, т. е. такое целое число N, что N≤X<N+1 Например: INT(3.7) = 3 INT(-3.7)= -4
LOG(X)	Дает натуральный логарифм X; аргумент X должен быть положительным. Для перевода в обычный логарифм (по основанию 10) пользуйтесь соотношением обычный логарифм (X) =0.434294481*натуральный логарифм (X)
RND или RND(X)	В форме RND функции никакого аргумента не передается. Эта функция возвращает следующее значение из последовательности псевдослучайных чисел, равномерно распределенных в диапазоне 0... 1. (См. ниже оператор RANDOMIZE.) В форме RND(X) аргумент X может служить фиктивным параметром или же использоваться для выбора другой последовательности случайных чисел. Детали применения в различных системах см. в приведенных далее таблицах
SGN(X)	Дает знак X согласно следующим правилам: -1, если X < 0; 0, если X = 0; +1, если X > 0,
SIN(X)	Дает синус X, где X задается в радианах (см. также ATN, COS)

SQR(X)	Дает неотрицательное значение квадратного корня из X; аргумент X должен быть положительным или равным 0
TAN(X)	Дает тангенс X, где X задается в радианах (см. также SIN, ATN, COS)

Примечание. RANDOMIZE N, где N — константа или переменная представляет собой оператор, а не функцию и используется для того, чтобы функция RND выдавала другую последовательность случайных чисел.

При пользовании функциями из табл. 3.1 могут быть полезны следующие тригонометрические тождества:

tg(x)=sin(x)/cos(x) ctg(x)=cosx(x)/sin(x)

$$\sec(x) = \frac{1}{\cos(x)} \qquad \operatorname{cosec}(x) = \frac{1}{\sin(x)}$$

$$\operatorname{sh}(x) = \frac{e^{x} - e^{-x}}{2} \qquad \operatorname{ch}(x) = \frac{e^{x} + e^{-x}}{2}$$

$$\operatorname{th}(x) = \frac{\operatorname{sh}(x)}{\operatorname{ch}(x)}$$

Так как из обратных тригонометрических функций в число стандартных входит только ATN (X) [arctg (x)], то другие функции можно получить из соотношений

$$\arcsin(x) = 2\operatorname{arctg}\left[\frac{1-\sqrt{(1-x^2)}}{x}\right]$$

arccos (x)=π/2 - arcsin (x)

Приведенная ниже программа иллюстрирует применение нескольких стандартных функций:

```
10 REM ПРИМЕНЕНИЕ ФУНКЦИЙ
20 PRINT "X"," ABS(X)","INT(X)","SGN(X)"
30 FOR X=-1.5 TO 1.5 STEP .5
40   PRINT X, ABS(X), INT(X), SGN(X)
50 NEXT X
60 END
```

RUN

X	ABS(X)	INT(X)	SGN(X)
-1.5	1.5	-2	-1
-1	1	-1	-1
-.5	.5	-1	-1
	б	0	0
.5	.5	0	1
1	1	1	1
1 .5	1.5	1	1

END AT LINE
60

Чтобы получить требуемый формат вывода при работе с Бейсиком BBC, модифицируйте строку 20 так, чтобы в ней была указана одна строка символов с соответствующим числом пробелов между заголовками:

```
20  PRINT "X    ABS (X)   INT (X)    SGN (X)"
```

Во многих системах с Бейсиком предусмотрены дополнительные функции; в табл. 3.2 — 3.4 приводятся функции, предусматриваемые некоторыми из наиболее популярных версий Бейсика.

Таблица 3.2. *Числовые функции, предлагаемые в Бейсике для персональной ЭВМ* ZX81 *фирмы* Sinclair *дополнительно к перечисленным в табл. 3.1 стандартным функциям*

Функция	Действие
ACS(X)	Дает арккосинус X, т. е. выражаемое в радианах значение угла, косинус которого равен X. (См. ATN в табл. 3.1.)
ASN(X)	Дает арксинус X, т. е. выражаемое в радианах значение угла, синус которого равен X. (См. ATN в табл. 3.1.)
LOG	Такой функции в данной версии Бейсика нет; ее заменяет функция LN
LN(X)	Дает натуральный логарифм X. (См. LOG в табл. 3.1.)
π	Эта функция не имеет аргумента. Дает значение константы π =3.141592654
RND	Дает значение следующего представителя последовательности случайных чисел. Аргументов не имеет. Оператор RAND N обеспечивает переход к новой последовательности случайных чисел

Таблица 3.3. *Числовые функции, предлагаемые в Бейсике* Microsoft *дополнительно к перечисленным в табл. 3.1 стандартным функциям*

Функция	Действие
CDBL(X)	Преобразует стандартное вещественное значение X в значение двойной точности. Например: в результате исполнения оператора A# =CDBL (454.67) переменная A# получит значение представления числа 454.67 с повышенной точностью
CINT(X)	Преобразует стандартное вещественное значение X в целое число (вызывает те же действия, что и функция INT). Учтите, что при этом X должно быть в диапазоне +32767
CSNG(X#)	Преобразует значение двойной точности X# в стандартное вещественное значение
FIX(X)	Отбрасывает дробную часть у числа X. Эквивалентна выражению SGN(X)* INT(ABS(X)) и совпадает с INT(X) при положительном X
RND(X) или RND	Дает следующее случайное число со значением между 0 и 1. При X < 0 обеспечивает переход к новой последовательности; при X=0 вновь дает значение предыдущего случайного числа. Если X > 0 или опущено, то дает значение следующего случайного числа. Для перехода к новой последовательности случайных чисел можно пользоваться также оператором RANDOMIZE N

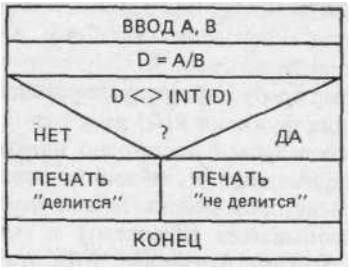
Таблица 3. 4. *Числовые функции, предлагаемые в Бейсике BBC дополнительно к перечисленным в табл. 3.1 функциям*

Функция	Действие
ACS(X)	Дает арккосинус X, т. е. выражаемое в радианах значение угла, косинус которого равен X (См. ATN в табл. 3.1)
ASN(X)	Дает арксинус X, т. е. выражаемое в радианах значение угла, синус которого равен X (См. ATN в табл. 3.1)
DEG(X)	Переводит значения X из радиан в градусы
LN(X)	Дает натуральный логарифм X. Эквивалентна функции LOG из табл. 3.1
LOG(X)	Дает обычный (десятичный) логарифм X. Обратите внимание на это отличие от стандарта, приведенного в табл. 3.1
RAD(X)	Переводит значения X из градусов в радианы
RND (1)	Дает следующее случайное число со значением между 0.0 и 0.999999
RND (0)	Дает значение предыдущего случайного числа, полученного с помощью RND (1)
RND(-K)	Отрицательное значение аргумента вызывает переход к новой последовательности случайных чисел (как в случае оператора RANDOMIZE) и возврат этого значения в качестве результата
RND(N)	При положительном целом N дает случайное целое число в диапазоне 1 .. .N
RND	При отсутствии аргумента возвращает случайное целое число в диапазоне ±2147483647 (диапазон машинного представления целых чисел 32 битами)

В приведенной ниже программе для проверки, делится ли одно число на другое нацело, используется функция INT (целая часть). Если деление нацело имеет место, то N1/N2=целая часть (N1/N2) :

```
10 INPUT A,B
20 D=A/B
30 IF INT(D)<>D THEN 60
40   PRINT A; "ДЕЛИТСЯ НАЦЕЛО НА" ;B
50   GOTO 70
60   PRINT A; "НЕ ДЕЛИТСЯ НАЦЕЛО НА";B
70 END
RUN
?96.12
96 ДЕЛИТСЯ НАЦЕЛО НА 12
END AT LINE 70
RUN
?21.4
21 НЕ ДЕЛИТСЯ НАЦЕЛО НА 4
END AT LINE 70
66
```

Структограмма этой программы такова:



Простые числа уже много лет являются объектом исследования. Обсуждавшаяся выше программа предоставляет определенные возможности для решения вопроса, является ли данное число простым или нет. Простым называется такое целое (положительное) число, у которого нет других делителей, кроме единицы и самого себя. Следующая структограмма иллюстрирует программу, определяющую, является ли вводимое значение переменной A простым

Сама программа такова:

```
10  REM ПРОГРАММА ДЛЯ ПРОВЕРКИ ПРОСТОТЫ ЧИСЛА
га   INPUT A
30   FOR B=2 TO A-1
40       D=A/B
50       IF INT(D)<>D THEN 80
60           PRINT A; "НЕ ЯВЛЯЕТСЯ ПРОСТЫМ ЧИСЛОМ"
70           STOP
80   NEXT B
90  PRINT A; "ЯВЛЯЕТСЯ ПРОСТЫМ ЧИСЛОМ"
100   END
```

Хотя та часть, которая отвечает за проверку первого делителя, несколько изменилась по сравнению с предыдущей программой, тем не менее и структо-

грамма, и текст последней программы ясно показывают, как предыдущая программа используется в качестве блока внутри большой программы.

3.4.1. СЛУЧАЙНЫЕ ЧИСЛА

Генерация случайных чисел требует некоторых разъяснений, так как почти в каждой системе реализация функции RND имеет свои специфические особенности. Эта функция используется достаточно широко: многие игровые программы и эмуляторы применяют случайный порядок выбора элементов данных. ЭВМ не имеют генераторов действительно случайных чисел (такие генераторы должны реализовываться аппаратно) и генерируют числа программно, используя рекуррентные формулы. Хотя эти числа лишь псевдослучайные, обычно они образуют вполне приемлемые последовательности. Чаще всего случайные числа находятся в диапазоне 0. . . 1, но бывают и исключения. Например, в Бейсике BBC предусмотрено несколько вариантов (см. табл. 3.4).

Основная особенность функции RND состоит в том, что при каждом обращении к ней получается другое значение. Ниже приводится пример простой программы (в Вашей системе может получиться другая последовательность случайных чисел) :

```
10 REM ПРОСТЫЕ СЛУЧАЙНЫЕ ЧИСЛА
20 FOR I=1 TO 8
30   X=RND
40   PRINT I,X
50 NEXT I
60 END
```

```
RUN
1             0.586078
2             0.966241
3             0.579484
4             0.437442
5             0.191831
6             0.799346
7             0.617701
8             0.109323
END AT LINE 60
```

При работе с Бейсиком BBC для получения нужного формата вывода замените строку 40 на оператор

```
40   PRINT ; I,X
```

Так как числа генерируются по формуле, всегда начинающей с одного и того же стартового значения, то при каждом запуске программы командой RUN получится одна и та же последовательность. Исключение составляет Бейсик BBC, при работе с которым последовательность случайных чисел продолжается до тех пор, пока не будет инициирована заново. Хотя возобновление последовательности случайных чисел при каждом запуске программы и удобно для тестирования и отладки программы, оно мало приемлемо при эксплуатации программы. Обычно алгоритм генерации случайных чисел

опирается на стартовое "порождающее" значение, которое может быть изменено оператором RANDOMIZE, или вызовом RND (X) с отрицательным значением аргумента X, или каким-либо иным способом (см. табл. 3.1 — 3.4). После этого генерируется новая последовательность псевдослучайных чисел. Ниже приводятся примеры инициации новой последовательности для различных систем:

```
100   RANDOMIZE 92 100   RAND 4 100  X=RND (-42)
```

Если добавить в приведенный выше простой пример ввод значения и переустановку с его помощью генератора случайных чисел, то получатся следующие результаты:

```
10 REM ПРОСТЫЕ СЛУЧАЙНЫЕ ЧИСЛА
15 INPUT B
17 RANDOMIZE B
20 FOR I=1 TO 8
30   X=RND
40   PRINT I,X
50 NEXT I
60 END

RUN
?12
1             0.25
2             0.770436
3             0.663104
4             0.612085
5             0.402334
6             0.576949
7             0.964517
8             0.720529
END AT LINE 60
```

3.5. ФОРМАТ ВЫВОДА ДАННЫХ

В этом разделе речь пойдет о тех возможностях управления форматом вывода данных, которых вполне достаточно при написании программы для работы в режиме диалога. Если же требуется печатать важные таблицы и отчеты, то дополнительно могут потребоваться средства управления шириной столбцов и форматами вывода чисел. Такие возможности предоставляются оператором PRINT USING, обсуждаемым в разд. 6.3. Похожие возможности управления обеспечиваются в Бейсике BBC и другим путем (см. следующий подраздел).

3.5.1. СНОВА ОБ ОПЕРАТОРЕ PRINT

Начальные сведения об операторе PRINT приводились в подразд. 1.2.3. Оператор PRINT может изображать как текст, так и числа на устройстве вывода, которым обычно служит ВТУ. Текст может появляться в операторе PRINT либо в виде строковой константы, например:

```
10  PRINT "РЕЗУЛЬТАТ РАВЕН", A
```


либо в виде строковой переменной, например:

```
10   A$ = "ИТОГОВАЯ СУММА"
20 PRINT A$; B
```

Элементы списка данных, выдаваемых оператором PRINT, могут отделяться друг от друга запятой или точкой с запятой.

Если разделителем служит точка с запятой, то значения элементов данных печатаются вплотную. Чтобы соседние числа не сливались в одно число, после каждого числового значения дополнительно выводится пробел, а перед числом может быть выведен знак минус или пробел. Например, если A = 1, B = 2, C = -3, то оператор

```
10  PRINT "НАЧАЛО"; A; B; C; "КОНЕЦ"
```

изобразит строку следующего вида:

```
НАЧАЛО 1 2 -3 КОНЕЦ
```

Учтите, что в Бейсике BBC числовые значения не окаймляются пробелами, поэтому при работе с ним указанный оператор изобразит строку НАЧАЛО12-3КОНЕЦ

При разделении точкой с запятой к строковым значениям не добавляются ни ведущий, ни замыкающий пробелы. Поэтому оператор

```
10  PRINT "НАЧАЛО"; "КОНЕЦ"
```

изобразит строку НАЧАЛОКОНЕЦ

Если же в качестве разделителя используется запятая, то данные распределяются по зонам экрана ВТУ. Обычно экран разбивается на вертикальные зоны шириной от 14 или 15 символов и запятая вызывает печатание как строкового, так и числового значения от начала следующей зоны. Если последняя зона строки уже была заполнена, то происходит переход на следующую строку. Таким образом, оператор

```
10  PRINT A,B,C
```

изобразит строку

```
1               2               -3
```

Учтите, что Бейсик BBC отличается от других версий тем, что выравнивает строковые значения по левому краю зоны, а числовые значения — по правому. Поэтому при работе с ним указанный оператор изобразит строку

```
1               2               -3
```

Такой подход имеет определенное преимущество, так как при выводе нескольких строк числовые значения располагаются в столбцах более привычным образом, но при этом смежные значения могут сливаться.

Можно использовать любую комбинацию разделителей; все нижеперечисленные операторы допустимы:


```
100 PRINT A;B;C
100 PRINT A,B;C
100  PRINT A;B,C
```

Указание оператора PRINT самого по себе, без списка переменных, вызывает переход на новую строку.

Напротив, если в операторе PRINT список переменных завершается запятой или точкой с запятой, то следующий оператор продолжит выдачу на той же самой строке. Например, операторы

```
10  PRINT A;"ПЛЮС";
11  20   PRINT B; "РАВНО";
12  30   PRINT A+B
```

изобразят строку
1 ПЛЮС 2 РАВНО 3
Следующая программа использует описанные выше возможности для изображения гистограммы по точкам графика функции y $=x^2 + 3$:

```
5   REM ПРОГРАММА ДЛЯ ИЗОБРАЖЕНИЯ ГИСТОГРАММЫ
10   PRINT "--------------------ОСЬ Y--------------------"
20   FOR X=0 TO 7 STEP .5
30   Y=X**2+3
30   PRINT "!";
40   FOR H=l TO INT(Y)
44    PRINT "*";
46   NEXT H
48   PRINT
50   NEXT X
60   END
```

```
RUN
--------------------ОСЬ Y--------------------
!***
!***
!****
!*****
!********
!**********
!*************
!****************
!********************
!************************
!****************************
!********************************
!************************************
!****************************************
!********************************************
END AT LINE 60
```

Выдача каждой строки начинается оператором PRINT в строке 38, изображающим вертикальную черту. Затем с помощью цикла в строках 40 — 46 на той же строке изображается заданное число звездочек (равное целой части значения Y).
Оператор PRINT в строке 48 требуется для перехода к новой строке, отвечающей следующему значению X. В приведенном примере предполагается,

что после заполнения всех строк экрана выдача новой строки вызывает перемещение всего изображения вверх на одну строку, и выдаваемая строка изображается в самом низу экрана (режим построчного движения изображения) . В некоторых системах для выбора такого режима требуется предварительно нажать на определенную клавишу (управляющую переходом к режиму постраничной выдачи и обратно) или дать специальную команду системе с Бейсиком. (См. в табл. 3.5 описание команды SCROLL системы с Бейсиком для микроЭВМ ZX81 фирмы Sinclair; при работе с ней эту команду надо добавить строкой 35 к приведенной выше программе.) Разделители **элементов списка выводимых данных** *Общая форма записи:* , *или* ;
(,) *Запятая вызывает переход к следующей зоне для печатания значения элемента (т. е. вывод выравнивается по левому краю зоны, однако в Бейсике* BBC *это не так).*
(;) *Точка с запятой вызывает печатание значения элемента вплотную к предшествующему значению. Если список выводимых данных завершается запятой или точкой с запятой, то перехода к новой строке не происходит и следующий оператор* PRINT *продолжит изображение на той же строке.*
Оператор PRINT *с пустым списком выводимых данных (т. е. в нем не указано ни одного элемента) вызывает переход к новой строке.*
Зоны обычно фиксированы и, как правило, имеют ширину в 15 символов.
Замечания
(a) *В Бейсике* Sinclair *экран разбивается на строки по 32 позиции (пронумерованные от 0 до 31), а зоны начинаются в столбцах 0 и 16.*

(б) *Версии Бейсика* Microsoft *и* BBC *позволяют переустановить длину строки с помощью оператора* WIDTH N, *где* N - *константа или выражение, задающее число позиций в строке.*
(в) *В Бейсике* BBC *строковые данные выравниваются по левому краю зоны, а числовые — по правому.*
В Бейсике BBC *имеются дополнительные возможности:*
(') *Разделитель в виде апострофа вызывает переход на новую строку.*
Целый спектр возможностей предоставляется посредством присваивания значений четырехбайтовой целой переменной @%. Значение самого младшего байта (от 0 до 255) задает ширину зоны. Остальные три байта (справа налево) управляют числом печатаемых цифр (от 0 до 9), форматом печати числовых значений (0,1 или 2), а также результатом обращения к функции STR$
Значение каждого байта задается двузначным шестнадцатеричным числом. Таким образом, диапазон значений 00... FF. Приведем три примера, задав значение байта зоны и положив остальные три байта нулевыми:
@%=&0000 0009 *устанавливает ширину зоны равной 9,* @%=&00 00 00 0C *устанавливает ширину зоны равной 12,*

@%=&000000 14 *устанавливает ширину зоны равной 2ф. (Символ & указывает на запись числа в шестнадцатеричной форме.)*
Во многих системах с помощью оператора PRINT можно получить такие возможности управления размещением изображения на экране, как адресация курсора и псевдографика (см. разд. 6.6) .
3.5.2. ВЫВОД В ОПЕРАТОРЕ INPUT
Весьма часто бывает так, что при запросе программой ввода на экране изображается какое-то сообщение, указывающее вид вводимого значения или диапазон допустимых значений. Например, операторы

```
10   PRINT "ЧИСЛО ЛЕТ ="
20   INPUT Y
```

приведут к возникновению изображения
ЧИСЛО ЛЕТ = ?_
и при наборе вводимого значения его изображение будет появляться справа от вопросительного знака. Зачастую гораздо удобнее и культурнее, если изображение вводимого значения будет возникать непосредственно после текста сообщения, и такого эффекта можно добиться, указав в операторе PRINT после текста точку с запятой. Например операторы

```
10  PRINT "ЧИСЛО ЛЕТ =";
20   INPUT Y
```

приведут к возникновению изображения ЧИСЛО ЛЕТ =?_
и изображение набираемого значения будет появляться после вопросительного знака на одной строке с текстом.
Во многих системах для микроЭВМ допускается расширение возможностей оператора INPUT, имеющее тот же эффект, что проиллюстрирован в последнем примере. Общая форма записи расширения такова:
INPUT "текст"; список элементов данных Таким образом, оператор

```
10  INPUT "ЧИСЛО ЛЕТ =";Y
```

выдаст то же, что и выше: "ЧИСЛО ЛЕТ =?_". Бейсик BBC предоставляет больше возможностей, чем большинство других систем, и позволяет в списке элементов данных оператора INPUT смешивать текст и вводимые данные, отделяя текст от вводимых данных запятой или вовсе не указывая разделителя между ними, например:

```
10   INPUT "ЧИСЛО ЛЕТ =",Y,"И ПРОЦЕНТ =" I
```

См. подразд. 4.1.6 по поводу других особенностей применения запятой или точки с запятой при вводе оператором INPUT списка значений в одной строке.

3.5.3. ФУНКЦИИ, УПРАВЛЯЮЩИЕ ОПЕРАТОРОМ PRINT

До сих пор расположение текста и результатов управлялось употреблением в операторах запятых или точек с запятыми. Однако большие возможности управления обеспечиваются за счет применения таких функций, как TAB и SPC.

Функция TAB не является просто генератором определенного числа пробелов; она предпринимает попытку перемещения к столбцу, номер которого передан ей в качестве аргумента. Эта функция реализована для большинства версий Бейсика. Если ей в качестве аргумента передано значение N, то она пытается переместить текущую позицию печати следующего символа вперед (слева направо) в столбец с номером N. Если это невозможно из-за того, что текущая позиция превышает N, то функция осуществляет переход на новую строку и затем позиционирует вывод в столбец N новой строки. Таким образом, оператор

```
PRINT A;TAB (12) ;B;TAB(24); C
```

напечатает значение A, начиная с первой позиции строки, значение B — с двенадцатой, а значение C — с двадцать четвертой.

Так как функция TAB не выдает пробелы, а перемещает позицию печати в заданный столбец, то она представляет собой идеальное средство для выдачи правильным образом оформленных таблиц значений, если число позиций, занимаемых под изображение этих значений, переменно. Например, в программе агента по торговле недвижимостью для выдачи цен жилищ могли бы появиться следующие строки:

```
10  PRINT "ГОРОДСКАЯ КВАРТИРА"; TAB (25) ; P1 20  PRINT "СМЕЖНЫЙ ДОМ"; TAB (25) ; P2
30  PRINT "ОСОБНЯК"; TAB (25) ; P3 40  PRINT "САРАЙ, ТРЕБУЮЩИЙ УХОДА"; TAB (25) ; P4
```

В результате цены будут выстроены в столбик независимо от длины описания вида жилища:

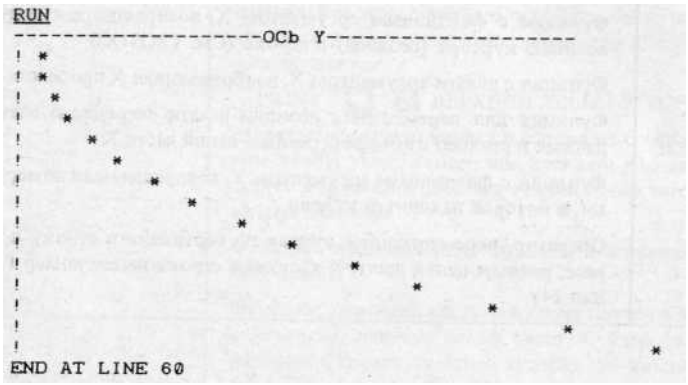
```
ГОРОДСКАЯ КВАРТИРА          1000
СМЕЖНЫЙ ДОМ                 2000
ОСОБНЯК                    3000
САРАЙ, ТРЕБУЮЩИЙ УХОДА     200
```

Реализации функции TAB различаются. По крайней мере, в одной — двух системах упомянутого выше перехода на новую строку не происходит, и если текущая позиция печати превышает N, то функция TAB попросту игнорируется.

Используя функцию TAB со значением аргумента X, вычисляемым в программе, можно получать грубые изображения графиков. В качестве X можно задавать любое выражение; его дробная часть отбрасывается, так как при изображении возможны только целочисленные координаты. Программу выдачи гистограммы из подразд. 3.5.1 нетрудно модифицировать следующим образом:


```
5 REM ПРОГРАММА ДЛЯ ГРУБОГО ПОСТРОЕНИЯ ГРАФИКА
10 PRINT "----------------------ОСЬ Y--------------"
20 FOR X=0 TO 7 STEP .5
30   Y=X**2+3
40   PRINT "!" ;TAB(Y) ;'*"
50 NEXT X
60 END
```

Эта программа выдаст следующее изображение (см. замечания о перемещении изображения при заполнении экрана, подразд. 3.5.1) :



Существуют другие полезные функции управления выводом, не входящие в стандарт Бейсика. В табл. 3.5 — 3.8 перечислены функции управления выводом, имеющиеся в некоторых системах. Обратите внимание на функцию SPC(N), выводящую N пробелов, где N ограничено диапазоном 0... 255.

Некоторые другие функции позволяют перемещать курсор (т. е. текущую позицию печати) назад, вперед, на другие строки экрана, а также выдавать новое изображение на месте уже существующего.

Таблица 3. 5. *Функции управления выводом в Бейсике для персональной ЭВМ* ZX81 *фирмы* Sinclair

Функция/оператор	Назначение
ATX, Y	Не является вещественной функцией, а применяется в операторе PRINT для указания строки (Y) и столбца (X) , в которых должно начаться изображение следующего элемента данных (Верхняя строка экрана имеет номер 0, нижняя 21, крайний левый столбец строки имеет номер 0, крайний правый 31)
CLS	Оператор, вызывающий стирание изображения с экрана
SCROLL	Оператор для перемещения изображения вверх на одну строку
TAB Y	Не является вещественной функцией, а применяется для перемещения позиции печати очередного элемента данных в столбец Y

Таблица 3.6. *Функции управления выводом в Бейсике* Microsoft

Функция/оператор	Назначение
HOME	Оператор, вызывающий стирание экрана и перемещение курсора в левую верхнюю позицию экрана
HTABX	Оператор, перемещающий курсор в столбец, номер которого равен целой части X (крайний левый столбец имеет номер 1, крайний правый 40) (См. VTAB)
POS(X)	Функция с фиктивным аргументом X, возвращающая текущую позицию курсора (столбец) в строке (См. VPOS(X))
SPC(X)	Функция с целым аргументом X, изображающая X пробелов
TAB (X)	Функция для перемещения позиции печати

	очередного элемента данных в столбец с номером, равным целой части X
VPOS(X)	Функция с фиктивным аргументом X, возвращающая номер строки, в которой находится курсор
VTAB X	Оператор, перемещающий курсор по вертикали в строку с номером, равным целой части X (верхняя строка имеет номер 1, нижняя 24)

Таблица 3. 7. *Функции управления выводом в Бейсике* BBC

Функция/оператор	Назначение
CLS	Оператор, вызывающий стирание экрана и перемещение курсора в левую верхнюю позицию экрана (с координатами (0,0))
COUNT	Функция без аргументов (входных параметров), возвращающая число символов, напечатанных на текущей строке
POS	Функция без аргументов, возвращающая горизонтальную позицию курсора (крайний левый столбец имеет номер 0)
SPC(X)	Функция, изображающая пробелы в количестве, равном целой части аргумента X
TAB (X)	Функция, перемещающая позицию печати очередного элемента данных в столбец с номером, равным целой части X
TAB(X,Y)	Функция, вызывающая перемещение курсора в позицию с координатами (X, Y) (см. POS, VPOS)
VPOS	Функция без аргументов, возвращающая вертикальную позицию курсора (верхняя строка имеет номер 0)

Таблица 3.8. *Функции управления выводом в Бейсике для персональной ЭВМ* PET *фирмы* Commodore

Функция/оператор	Назначение
10 PRINT "текст", A; B	Обычный оператор PRINT, с помощью которого можно выводить текст и числовые значения. Однако если при наборе заключенного в кавычки текста нажимается какая-либо из клавиш управления курсором, то генерируется специальный символ, а соответствующее перемещение курсора происходит во время исполнения этого оператора в программе. Например, оператор PRINT "♥ S ВЕРХНИЙ ЛЕВЫЙ УГОЛ" сотрет содержимое экрана и изобразит сообщение в его верхнем левом углу. Учтите, что клавиши управления курсором станут нормально работать только после того, как будут закрыты кавычки
SPC(X)	Функция, изображающая пробелы в количестве, равном целой части X

TAB (X)	Функция, перемещающая текущую позицию печати в столбец с номером, равным целой части X. Если текущая позиция находится справа от этого столбца, то никаких действий не предпринимается

Кроме этих функций в системе может присутствовать функция CLS, вызывающая стирание экрана и перемещение курсора в его левый верхний угол. Взятые вместе, эти функции дают программе на Бейсике полную свободу управления позиционированием вывода в любое место экрана ВТУ. Эти функции могут применяться для "оживления" псевдографического вывода в игровых программах, в прикладных программах, демонстрирующих, к примеру, поток жидкости по трубам и т. д. По поводу псевдографики см. разд. 6.6.

3.6. НАЧАЛЬНАЯ СТАДИЯ РАЗРАБОТКИ ПРОГРАММЫ

Вам может показаться, что если Вы познакомились с операторами, составляющими язык программирования, то, чтобы стать квалифицированным программистом, не хватает лишь накопить опыт. В каком-то смысле это так, поскольку в процессе изучения языка Вам пришлось, рассматривая примеры и решая упражнения, неявным образом усвоить и определенные знания о способах разработки программ. В действительности же доскональное знание методов разработки составляет основную часть искусства квалифицированного программиста. При необходимости оно может быть использовано при решении задач с помощью самых разных языков программирования. Поэтому правильные методы разработки слишком важны для того, чтобы ограничиваться их усвоением лишь на примерах. Современные методы разработки программ обсуждаются в нескольких главах этой книги.

Прежде всего, для чего требуется программирование? Основной его целью является решение задач и создание работающих систем. И не просто работающих, а работающих правильно, и притом все время. Самая худшая ситуация, которую только можно представить, — это когда кажется, что программа работает, и она действительно тщательно тестирована, но при всем том, к примеру, в одном из тысячи исполнений выдает неверные результаты. Пользователи и авторы такой программы склонны доверять ей и могут дорого поплатиться за вдруг возникшую ошибку в результатах. Поэтому правильность является первым критерием качества программы.

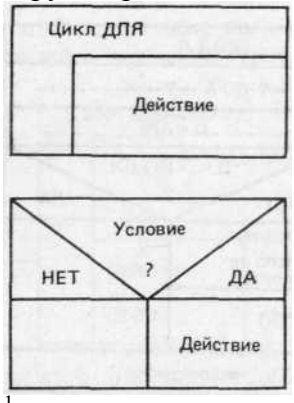
Задачи меняются и развиваются, изменяются данные, а также аппаратная часть ЭВМ. Поэтому даже бесхитростная программа может нуждаться в изменениях в течение своего жизненного цикла. Изменения не исключение, а норма для программируемой системы. Поэтому следующее свойство, которого надо добиваться при разработке программы, — простота модификации.

Бурный прогресс микроэлектроники привел к тому, что стоимость аппаратной части ЭВМ стала гораздо ниже стоимости разработки программного обеспечения. В 50-х и 60-х годах это соотношение было обратным; поэтому в то время появилось много "хитрых" методов программирования, направленных на экономию памяти ЭВМ, сокращение времени исполнения программы, минимизацию числа обращений к диску. К сожалению, эти ухищрения были потенциальным источником ошибок; программы было трудно читать и почти невозможно модифицировать. Вывод ясен: при разработке программ надо добиваться их эффективности всеми средствами, но не любой ценой. Достижение правильности программы и простоты ее модификации важнее достижения эффективности. Но как же добиться этих похвальных целей? До сих пор основным производителем программ является человек, мышление которого индивидуально и иррационально. Поэтому любая оценка методов работы программистов с точки зрения того, насколько эффективно достигаются указанные выше цели, сопряжена с большими трудностями. Однако в настоящее время многие специалисты в области программирования пропагандируют метод разработки программ, известный под названием структурного проектирования. Проводившиеся сравнения структурного проектирования с другими методами показали, что оно имеет заметные преимущества.

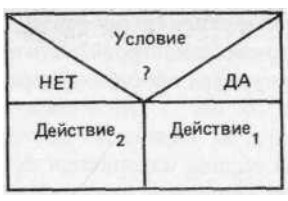
В настоящее время существует несколько различных методов, относящихся к категории структурного проектирования. Один из них применим к задачам, в которых приходится иметь дело с

большими объемами структурированных данных. По этому методу структура программы определяется исходя из структуры данных. Другой метод, которому мы собираемся следовать в этой книге, начинается с описания решения задачи в самом общем виде. Затем в это описание шаг за шагом вносятся детали до тех пор, пока это описание не станет настолько подробным, что по нему будет достаточно легко написать программу. Этот метод часто называют "пошаговой детализацией", и так как процесс его применения начинается с написания общего представления решения задачи, а завершается написанием конкретной про-

граммы, то о нем говорят как о проектировании "сверху вниз". Все методы структурного проектирования построены по принципу ."сверху вниз". В разд. 1.3 была описана структура простой программы в следующем виде:
Ввод
Обработка
Вывод
отвечающая верхнему уровню разработки. Простая программа на Бейсике, приведенная в гл. 1, отвечает нижнему уровню разработки. В данной главе уже шла речь о нескольких новых элементах языка Бейсик, структурой которых можно воспользоваться при разработке программ, применив следующую форму записи[1]:
ДЛЯ цикла действие
СЛЕДУЮЩИЙ проход цикла
ЕСЛИ условие ТО действие
ЕСЛИ условие ТО действие$_1$ ИНАЧЕ действие$_2$
Здесь "действие" представляет собой один или несколько операторов, не вызывающих перехода в другую часть программы. Первые три элемента описаний, ввод, обработка и вывод, служат примерами действий. Трем только что введенным элементам описаний соответствуют следующие структограммы:



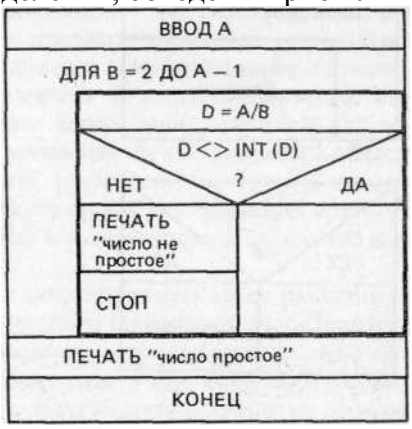
[1]В оригинале обыгрывается значение служебных слов Бейсика как слов английского языка. В настоящем переводе такое употребление передается набором русского эквивалента служебного слова прописными буквами: ДЛЯ соответствует служебному слову FOR, СЛЕДУЮЩИЙ - NEXT, ЕСЛИ - IF, ТО - THEN, ИНАЧЕ - ELSE, ВВОД -INPUT, ПЕЧАТЬ - PRINT, ПЕРЕЙТИ К - GO TO. - *Прим. перев.*



Конечно, действие может быть и структурой ДЛЯ-СЛЕДУЮЩИЙ (т. е. FOR-NEXT) или ЕСЛИ (IF). Примерами могут служить детально обсуждавшиеся в подразд 3.3.2 вложенные циклы и программа из подразд. 3.3.3, в которой структура ЕСЛИ (IF) была вложена в цикл ДЛЯ-СЛЕДУЮЩИЙ (FOR-NEXT).

В терминах структограмм действие представляется прямоугольником, и, как уже было показано в этой главе, все до сих пор описанные программные конструкции внешне представляют собой прямоугольники. Поэтому их очень легко вкладывать друг в друга или смыкать друг с другом.

Указываемый в структограмме прямоугольник действий можно рассматривать как модуль, т. е. группу действий, логически образующих единое целое. Принцип модульности иллюстрируется программой деления нацело и программой распознавания простых чисел из разд. 3.4: после небольших изменений программа деления образовала модуль, помещенный в программу распознавания простых чисел. На приведенной ниже полной структограмме программы распознавания простых чисел внутренний модуль, образованный слегка измененной программой деления, обведен жирной линией:



Как видно из этой структограммы, модуль образован не одним действием, а совокупностью действий, составляющих логически более или менее завершенную группу.

Продемонстрируем метод пошаговой детализации на примере решения задачи распознавания простых чисел. Прежде всего надо описать общий

метод: требуется выяснить, имеет ли вводимое значение какие-либо делители, кроме единицы и самого себя. Такое описание называют первым уровнем разработки.

Уровень 1 Ввести число
Проверить, имеет ли оно делители Вывести результат

Составные части уровня 1 рассматриваются с целью расширения, другими словами, детализации. Детализируется любая составная часть, которая не может быть непосредственно выражена в виде операторов Бейсика. Структограмма этого уровня такова:

Ввод
Проверка
Вывод

Детализация 1.1: проверить, имеет ли число делители. ДЛЯ каждого возможного делителя введенного числа ЕСЛИ введенное число делится нацело на делитель ТО
ПЕЧАТЬ "число не простое" и СТОП СЛЕДУЮЩИЙ делитель

Структограмма этой детализации представляет собой расширение центрального прямоугольника "проверка" уровня 1.

Другие основные части описания не требуют детализации, но теперь после того, как мы определили в детализации 1.1, что выход из цикла ДЛЯ-СЛЕДУЮЩИЙ (FOR-NEXT) происходит после проверки всех возможных делителей в ситуации, когда введенное число ни на один из них не поделилось нацело, "вывод результата" можно расписать подробнее. После объединения всех частей и выделения (переводов) служебных слов Бейсика прописными буквами можно полностью составить следующий уровень описания программы:

Уровень 2 ВВОД числа
ДЛЯ каждого возможного делителя введенного числа ЕСЛИ введенное число делится нацело на делитель ТО
ПЕЧАТЬ "число не простое" и СТОП СЛЕДУЮЩИЙ делитель ПЕЧАТЬ "число простое" КОНЕЦ

Мы дошли уже почти до уровня операторов Бейсика. Единственное, что еще требует уточнения, это предложение, следующее за ЕСЛИ.

Детализация 2.1. ЕСЛИ введенное число делится нацело на делитель.

В терминах операторов Бейсика это предложение можно написать:

```
IF (A/B)=INT (A/B) THEN . ..
```

или более эффективно, избегая одного лишнего деления:

```
D=A/B
IF D=INT (D) THEN .. .
```

В последнем случае над слегка измененным прямоугольником ЕСЛИ в приведенной выше структограмме возникает прямоугольник процесса.

Теперь программу можно записать полностью на языке операторов Бейсика (см. разд. 3.4). В этом примере не было большого числа уровней, но он достаточно хорошо иллюстрирует метод разработки.

Этот метод получит развитие в последующих главах, но он заслуживает несколько слов и здесь.

При продемонстрированном выше подходе мы имели дело как со словесными описаниями, так и с диаграммами. Некоторые программисты пользуются только одной формой, например словесными описаниями. Вам тоже может больше подойти какая-то одна из этих двух форм.

Иногда структуры содержат оператор GO TO. Учтите, однако, что этому оператору не соответствует ни один символ структограммы. Поэтому придерживайтесь следующего правила: всеми силами старайтесь использовать оператор GO TO для образования правильных структур, но ни в коем случае не применяйте его произвольным образом.

Вероятно, Вы уже обращали внимание, что переложение на операторы Бейсика некоторых структур, например ЕСЛИ-ТО-ИНАЧЕ (IF-THEN-ELSE), выглядит довольно тяжеловато. Если это несколько тревожило Вас, то Вы были правы в своих ощущениях, поскольку Бейсик не ориентирован на структурное проектирование и пользоваться им надо аккуратно. Дополнительные средства, включенные в Бейсик в некоторых системах, помогают справиться с некоторыми трудностями в этой области.

Ни одна разработка не производится в чистом виде "сверху вниз". Всегда приходится кое-где забегать вперед и идти снизу вверх, предугадывая, что

же надо выполнить на нижних уровнях, и в конечном счете - на самом нижнем уровне, т. е. уровне языка программирования. Например, нет никакого смысла разрабатывать методом "сверху вниз" программы поиска данных и требовать от нее на нижнем уровне произвольного доступа к хранящимся в файле элементам данных, если такой метод доступа не обеспечивается Вашей системой. (Файлы и методы доступа к данным обсуждаются в гл. 8.)

УПРАЖНЕНИЯ

3.1. Стоимость поездки на такси определяется как сумма посадочной платы в 25 пенсов и стоимости пробега из расчета 8 пенсов за каждые полмили сверх двух миль. Пользуясь словесными описаниями или структограммами, разработайте программу, которая воспринимает в качестве ввода длину пробега такси в целых милях, а выводит плату за проезд.

Один из возможных ответов приведен в решениях упражнений в приложении 1. Переложите это или Ваше собственное решение на операторы Бейсика и запустите полученную программу. Что произойдет, если ввести не целое, а дробное число миль? Например, 2.4 и 2.7 мили? Возникающая в этих случаях проблема станет понятнее, если определить стоимость пробега в 8 пенсов за каждые полные полмили и последнюю незаконченную часть. В этой ситуации может оказаться полезной одна из числовых функций.

3.2. Следующая структограмма иллюстрирует очень простую программу, печатающую число, его квадрат и куб:

ВВОД N
ПЕЧАТЬ N, N*N, N*N*N
КОНЕЦ

Взяв эту структограмму за основу, нарисуйте структограмму, представляющую программу печати таблицы квадратов и кубов всех чисел от 1 до 10. Добавьте соответствующие заголовки столбцов, чтобы показать смысл каждого из них.

3.3. Доработайте программу из упражнения 3.2 так, чтобы на каждой строке печатались степени числа до пятой включительно. Используйте для этого дополнительный цикл FOR-NEXT и средства управления выводом. Иначе говоря, на каждой строке должны печататься N^I, где I изменяется от 1 до 5. При N =4 вывод должен выглядеть следующим образом:

4 16 64 256 1024

Добавьте к столбцам соответствующие заголовки. Столбцы могут оказаться плохо выравненными из-за разного количества цифр в числах. Для исправления этого дефекта можете попробовать применить имеющиеся в Вашей системе функции управления выводом.

3.4. Разработайте и напишите программу для вычисления факториала N, где N — целое число. Факториал часто записывается как N! и определяется соотношением

N!=1*2*3*4* ... *N

Таким образом,

2! =1*2=2

5!=1*2*3*4*5=120

Значение N должно вводиться в программу, от которой требуется напечатать это значение и его факториал, снабдив вывод соответствующими пояснениями. Факториал находит непосредственное применение во многих задачах. Например, если мы хотим сделать упорядоченную выборку R из N различных объектов, то число разных выборок (иногда называемых размещениями и обозначаемых N^PR) равно N!/(N-R)!. Учтите, что 0! по определению равно 1.

Пусть, к примеру, владелец магазина располагает восемью различными новыми книгами для оформления витрины в канун рождественских праздников, но сразу может выставить только пять из них. Тогда число разных вариантов оформления витрины равно числу размещений по 5 из 8, т. е. 8!/(8-5) ! =6720.

3.5. Разработайте и напишите программу для вычисления среднего значения оценок, выставленных на экзамене группе студентов. Оценка ставится в процентах от единицы и, следовательно, может принимать значения от 0 до 100. Программа должна быть настолько гибкой, чтобы ею воспринималось любое количество оценок. Один из способов достичь этого - вводить особое значение в качестве признака конца ввода группы оценок. Это особое значение не должно использоваться в вычислениях; чтобы его можно было распознать, оно должно лежать вне диапазона допустимых оценок. В нашем случае в качестве особого значения можно выбрать отрицательное число или же число, превышающее 100.

3.6. Разработайте и напишите программу, которая проверяет четность вводимого числа и выводит само это число и результат проверки (четное и нечетное) .

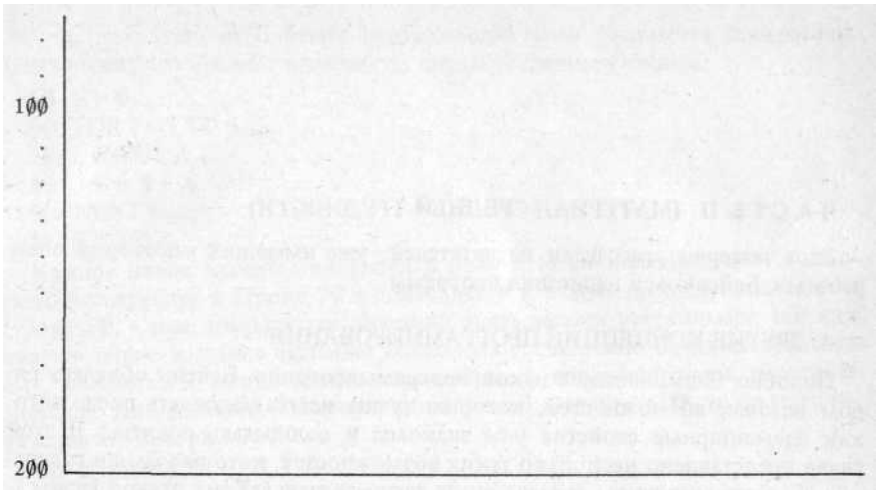
3.7. Разработайте и напишите программу, которая определяет наибольшее количество положительных четных чисел, сумма которых не превышает 100.

Чтобы программа выполняла в точности то, что требуется в условии задачи, над ней надо поразмыслить. Вам придется поупражняться как в программировании, так и в точности интерпретации условия задачи.

3.8. Разработайте и напишите программу печати таблицы, помогающей определять налог с оборота, начисляемый на проданные товары. Налог составляет 15 % от стоимости товара. Все величины должны выражаться в пенсах (100 пенсов =1 фунт) ; таблица должна быть составлена для цен от 0 пенсов до 2 фунтов стерлингов с шагом 2 пенса и должна иметь подобающий заголовок:

Налог с оборота для цен от 0 пенсов до 2 фунтов стерлингов

	0	2	4	6	8
0	0	0	0	0	1
10	1	1	2	2	2
20	3				
30					
40					
.		и т. д.			



Учтите, что дробная часть пенса отбрасывается и соблюдение заданного формата таблицы очень важно. В этой задаче много общего со структурами, используемыми в задаче 3.3.

Если Вы испытываете склонность к математике, то можете использовать подобные структуры для программы печати таблиц того же вида, который встречается в справочниках по тригонометрическим функциям, например для таблицы значений синуса углов 0 ... 90 градусов с шагом два градуса.

3.9. Разработайте и напишите программу для решения следующей задачи моделирования:

Поезд, на котором Вы ездите на работу, может отойти от станции с опережением расписания до 10 мин или с отставанием от расписания до 5 мин. Если Вы в течение месяца (равного 20 рабочим дням) приходите на станцию к моменту отправления поезда согласно расписанию, то сколько раз Вам удастся сесть на этот поезд?

Для моделирования реальной ситуации используйте генератор случайных чисел. Исходите из того, что все времена в целых минутах из указанного выше интервала равновероятны. Таким образом, всего возможно 15 отстоящих друг от друга на минуту времен отправления поезда, от 10-минутного опережения расписания до 5-минутного отставания. Вам же удастся сесть на поезд только в тех случаях, когда он отходит вовремя или опаздывает.

3.10. В одной из разновидностей игры в метание стрелок требуется набрать ровно 301 очко. Новичок бросает стрелку так, что она попадает в мишень, с равной вероятностью давая от 0 очков (мимо мишени) до 20. Разработайте и напишите программу для моделирования этой игры и подсчета числа метаний стрелки.

Исходите из того, что удвоения и утроения очков не бывает. Чтобы достигалась сумма ровно в 301 очко, результат очередного метания не учитывается, если сумма должна превзойти 301, и метание продолжается, пока в результате очередного метания сумма очков не станет в точности равной 301.

ЧАСТЬ II (МАТЕРИАЛ СРЕДНЕЙ ТРУДНОСТИ)

Этот материал рассчитан на читателей, уже имеющих некоторый опыт работы с Бейсиком и написания программ.

4. ДРУГИЕ КОНЦЕПЦИИ ПРОГРАММИРОВАНИЯ

Подобно большинству языков программирования, Бейсик обладает рядом важных возможностей, которые лучше всего обсуждать после того, как элементарные свойства уже знакомы и полностью поняты. В этой главе представлено несколько таких возможностей, которые хотя и различны, но в определенном смысле очень тесно связаны. Тема данной главы — обсуждение объектов, над которыми проводятся вычисления, другими словами, "предметов", которыми манипулирует язык программирования.

До сих пор такими "предметами" в основном были числа в виде констант и значений числовых переменных. Если мы можем успешно манипулировать этими простыми предметами, то почему бы не ввести в рассмотрение более сложные предметы и, если они окажутся подходящими, не использовать их тем или иным образом? Бейсик позволяет простым образом пользоваться группами (или массивами, или списками) числовых значений. В нем также предусмотрены другой тип простых переменных, способных содержать символы, а не числа, и средства эффективного манипулирования этими строковыми переменными.

Хотя детали применения этих предметов могут быть в отдельных случаях довольно сложными, пусть это не затмит простоту сути описания и применения некоторых объектов.

Некоторые развитые языки программирования содержат средства определения Ваших собственных "предметов" (объектов или структур) и связей между ними, но эффективное их применение доступно только искусному программисту. Поработав с Бейсиком, Вы должны осознать его ограниченность в отношении подобных возможностей, после чего можно приступать к знакомству с другими языками.

4.1. МАССИВЫ

ЭВМ обладает тем достоинством, что при наличии соответствующей программы может очень быстро повторять последовательность операций столько раз, сколько нам требуется. Это свойство рассматривалось и развивалось в предыдущей главе на примере команд организации циклов. Напри-

мер, в результате исполнения приводимого ниже фрагмента программы будет вычислено среднее значение по шести введенным числам:

```
50   S = 0
60  FOR I = 1 TO 6
70      INPUT A
80      S = S+A
90  NEXT I 100   S = S/6
```

Каждое новое значение вводится в одну и ту же переменную (A) с помощью оператора в строке 70 и добавляется к сумме предыдущих чисел в строке 80. Сами предыдущие значения оказываются утерянными, так как каждое новое значение занимает место непосредственно предшествующего ему значения. В конце цикла переменная A содержит самое последнее значение. До настоящего момента единственно возможной была альтернатива написать программу

```
50   INPUT A, B, C, D, E, F
60   S = A + B + C + D + E + F
80   S = S/6
```

дающую тот же результат, но сохраняющую на будущее все шесть значений за счет использования шести переменных. Возникает впечатление, что при этом программа даже сокращается, но попробуйте написать программу вычисления среднего для 1000 значений!

В описанном выше случае, да и при многих других обстоятельствах нам надо иметь возможность манипулировать числами, не уничтожая их значений, подобно тому, как мы манипулировали простой переменной A в первом примере. Подходящим для этих целей объектом является массив, дающий общее имя группе ячеек памяти и обеспечивающий доступ к каждой отдельной ячейке путем указания индекса. На рис. 4.1 изображены несколько отдельных переменных, таких как A1 и N, а также массив по имени G. Это групповое имя, данное ячейкам, каждая из которых аналогична простой переменной. Первая переменная из этого массива имеет имя G(0), вторая G (1), третья G (2) и т. д. Образующие массив переменные называются элементами массива, а числа в скобках — индексами массива. Отдельными элементами массива можно пользоваться на тех же правах, что и простыми переменными. Рассмотрим фрагмент программы:

```
10  INPUT G(l)
20   G(2) =G(1)
30   PRINT G(1), G(2)
```

В строке 10 вводится значение; в строке 20 оно копируется, а в строке 30 оба значения выводятся. При этом G(l) и G(2) являются отдельными переменными. В точности те же самые действия произведет программа

```
10   INPUT Z
20  A=Z
30   PRINT Z, A
```

использующая простые переменные A и Z.

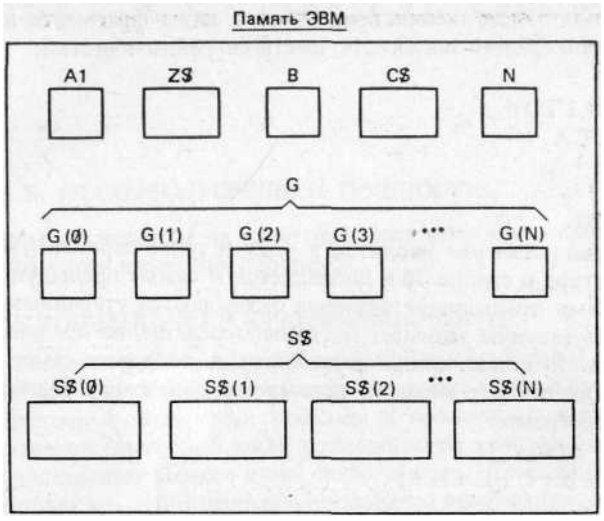


Рис. 4.1. Массивы в памяти ЭВМ

Нередко массивы используются для хранения родственных значений. Например, в массиве M можно регистрировать количество пинт молока, получаемых ежедневно. Пусть M(1) =1 для воскресной доставки, M(2) = 3, M(3) = 0, M(4) = 2, M(5) = 1, M(6) = 2 и M(7) =1. При таких значениях в массиве M оператор PRINT M (4) напечатает 2, но того же результата можно добиться с помощью операторов

```
100   D = 4
114   PRINT M(D)
```

Ценность массивов как раз и состоит в этой возможности использовать переменную в качестве индекса. Рассмотрим фрагмент программы:

```
50   FOR D = 1 TO 7
60     PRINT M(D)
70  NEXT D
```

Если в массиве M содержатся указанные выше значения, то при первом проходе цикла переменной D будет присвоено значение 1 и исполнится оператор PRINT M(l), на следующем проходе D будет присвоено значение 2 и исполнится PRINT M (2) , и т. д. Так как в конце оператора печати указана точка с запятой, то все значения будут изображаться на одной строке:

1 3 0 2 1 2 1

4.1.1. ОПЕРАТОР DIM

Прежде чем воспользоваться массивом, в программу надо включить оператор DIM, задающий максимально допустимый индекс. Это даст возмож-

ность системе с Бейсиком зарезервировать в памяти область достаточного размера. Например, оператор

```
10  DIM M(7)
```

зарезервирует область для хранения упоминавшегося выше массива.

При желании в одном операторе можно указать список массивов, например:

```
30  DIM A(14), M(7), Z(100)
```

Слово DIM представляет собой сокращение от DIMension (размер), поскольку с помощью этого оператора задается размер массива. Операторы DM можно указывать в любом месте программы на Бейсике, но лучше всего группировать операторы в самом начале программы, где их легче увидеть. Однако оператор DIM должен предшествовать упоминанию описанного в нем массива в других операторах программы.

Оператор DIM

Общая форма записи:

DIM *имя_массиеа (максимальный_индекс)*

"Имя_массива" обычно выбирается по тем же правилам, что и имена простых переменных. Допускаются как числовые массивы, так и массивы строк символов. В операторе можно указать список описаний массивов, разделяя их запятыми.

"Максимальный_индекс" указывает максимально допустимый в программе индекс и должен быть положительным. Иногда дополнительно требуется, чтобы он был целой константой. Индексы массива обычно пробегают значения от 0 до максимального, но в некоторых системах — от 1 до максимального (см. подразд. 4.1.2).

Можно пользоваться и многомерными массивами. Для этого в скобках после имени массива указывается несколько максимальных индексов. Число измерений обычно ограничено определенным пределом.

Если массив не описан в операторе DIM, *то по умолчанию ему приписывается максимальный индекс (10), если он одномерный, и (10,10), если он двумерный.*

Допускаются как числовые массивы, так и массивы строк символов. И те и другие могут быть одномерными или двумерными (большинство реализаций Бейсика допускает и большее число измерений). Если элемент массива используется раньше, чем встретится оператор DIM с описанием этого массива, то в большинстве систем такому массиву по умолчанию приписывается максимальный индекс (10), если массив одномерный, и (10,10), если массив двумерный. Не рекомендуется избегать описания всех используемых массивов в начале каждой программы и полагаться на значения максимальных индексов, приписываемых по умолчанию. Приведем несколько примеров описания массивов:

```
10  DIM A(50), Z(6)                  (числовые одномерные массивы)
20  DIM P$ (5)                       (одномерный массив строк)
30  DIM(C(10,5),D(10,20),E$(5,5)     (двумерные матрицы)
```

Прежде чем перейти к применению массивов, посмотрим внимательнее на их имена. Обычно они имеют тот же тип и ту же длину, что и имена простых переменных.
Если имя A9 допустимо для переменной, то допустимо и описание **DIM** Z2(10), т. е., как обычно, имя может состоять из буквы и цифры. Однако, по крайней мере, в одной системе для массивов допускаются только однобуквенные имена.
Если для переменной допустимо имя PINTOFMILK, то допустимо и описание DIM DAILYPINTA(7) .
Если допускаются целые переменные или переменные с двойной точностью, то допустимы и массивы этих типов. Не стесняйтесь выбирать одно и то же имя для простой переменной, массива или строковой переменной, так как система должна трактовать их как разные объекты. Таким образом, операторы

```
5   DIM A (4) 10   A = 10 20  A(0) =-9 30  A$="OK" 40  PRINT A,A(0),A$
```

должны выдать

```
10                    -9                    OK
```

Если же этого не происходит, тогда избегайте совпадения имен.
Коснемся теперь массивов с переменным размером. Бывает очень удобно использовать для решения задачи в точности необходимый (без запаса) размер массива. Например, для задания размера массива A во время исполнения программы можно воспользоваться операторами

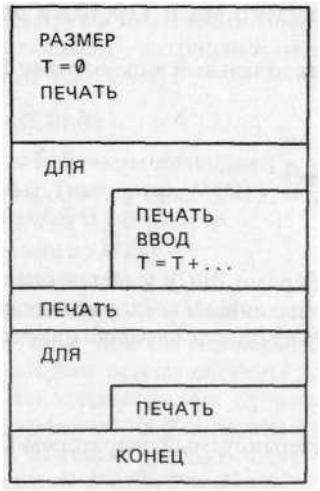
```
100  INPUT К 120 DIM A (К) 130  .....
```

В большинстве систем допускаются переменные размеры. При работе с теми системами, в которых этого нет, размер массива надо определять постоянным с расчетом на все допустимые случаи, несмотря на то, что в большинстве ситуаций при исполнении программы будет использоваться только часть массива.
Перейдем теперь к примеру работы с массивами. Пусть в системе управления запасами на складе хранятся товары 20 разных наименований, причем наименованиям приписаны номера от 1 до 20. Массив S используется для хранения в каждом его элементе текущего запаса товара соответствующего наименования. Приведенная ниже программа получает за счет ввода значение текущих запасов товаров, подсчитывает суммарный запас и печатает номер наименования, соответствующий запас в абсолютных единицах и процентах от суммарного запаса.
90

```
10 REM ИНВЕНТАРИЗАЦИОННАЯ ОПИСЬ
20 DIM S(20>
30 REM ВВЕДИТЕ 20 ТЕКУЩИХ ЗАПАСОВ ТОВАРОВ
40 REM И СРАЗУ ЖЕ ПОДСЧИТАЙТЕ ИХ ОБЩЕЕ КОЛИЧЕСТВО
50 T=0
60 PRINT "ПОЖАЛУЙСТА, ВВЕДИТЕ КОЛИЧЕСТВО ТОВАРА"
70 FOR I=1 TO 20
80   PRINT "ДЛЯ НОМЕНКЛАТУРНОГО НОМЕРА";I;
90  INPUT S(I)
100   T=T+S(I)
110 NEXT I
120 PRINT
130 PRINT "ВСЕГО";T;"ШТУК"
140 REM
150 REM ДЛЯ КАЖДОГО ТОВАРА НАПЕЧАТАЙТЕ ТРИ ВЫХОДНЫХ ЗНАЧЕНИЯ
160 PRINT "ТОВАР","ШТУК","В % ОТ СУММЫ"
170 FOR I=1 TO 20
```

```
180   PRINT I ,S(I) ,S(I)*100/T;"%"
190 NEXT I
200 END
```

Обратите внимание на то, что программа имеет очень простую структуру в виде двух отдельных циклов:



Если вы работаете с расширенным Бейсиком, то можете переписать программу следующим образом:

```
10 REM ИНВЕНТАРИЗАЦИОННАЯ ОПИСЬ
20 DIM STOCKITEM(20)
30 TOTAL=B
40 PRINT "ПОЖАЛУЙСТА. ВВЕДИТЕ КОЛИЧЕСТВО ТОВАРА"
50 FOR ITEM=1 TO 20 : REM ВВЕДИТЕ ТЕКУЩИЙ ЗАПАС ТОВАРА
60   PRINT "ДЛЯ НОМЕНКЛАТУРНОГО НОМЕР А" ; ITEM ;
70   INPUT STOCKITEM(ITEM)
80   TOTAL=TOTAL+STOCKITEM<ITEM) :REM ДОБАВЬТЕ К СУММЕ
90 NEXT ITEM
100 .....
91
```

и т. д., использовав содержательные имена для переменных и указав несколько операторов в одной строке (разделив их двоеточием) с тем, чтобы поместить комментарии в той строке, к которой они относятся.

4.1.2. ТРЕБУЕМАЯ ДЛЯ ХРАНЕНИЯ МАССИВОВ ПАМЯТЬ

В стандарте Бейсика (в минимальном подмножестве) предполагается, что индексы начинаются с 0, и это соглашение принято для большинства систем. В этом случае оператор DIM P(4) зарезервирует 5 ячеек:

P(0),P(1),P(2),P(3),P(4)

а оператор DIM A(2, 3) 12 ячеек:

A(0,0),A(0, 1), A(0,2), A(0,3) A(1,0), A(1,1), A(1,2), A(1,3)
A(2,0), A(2,1), A(2,2), A(2,3)

Так как для одномерного массива по умолчанию максимальный индекс равен (10), то для него будет зарезервировано 11 ячеек и индексы могут принимать значения от 0 до 10. Аналогично для двумерного массива по умолчанию максимальный индекс равен (10,10) и для индекса будет зарезервирована 121 ячейка.

В тех системах, в которых начальный индекс равен 1, оператор DIM P(4) зарезервирует 4 ячейки:

P(1),P(2),P(3),P(4) а оператор DIM A (2, 3) 6 ячеек:

A(1,1),A(1,2),A(1,3)

A (2, 1),A(2, 2),A(2, 3)
В большинстве систем (кроме BBC) имеется оператор OPTION BASE, позволяющий изменять устанавливаемое по умолчанию начальное значение индекса. Так, в системе, где начальное значение индекса первоначально равно 0, оператор
10 OPTION BASE 1
уменьшит объем памяти, резервируемый оператором DIM A (2, 3), с 12 до 6 ячеек.
Оператор OPTION
Общая форма записи:
OPTION BASE N *где* N *равно либо 0, либо 1.*
Устанавливает начальное значение индекса для всех используемых в программе массивов равным 0 или 1. В программе может быть не более одного оператора OPTION.
Выбор между 0 и 1 в качестве начального значения индекса должен определяться характером решаемой задачи. Скажем, в приведенном выше приме-

ре с разноской молока дни недели естественнее идентифицировать числами от 1 до 7, нежели от 0 до 6. Поэтому в этой задаче выбор 1 в качестве начального значения индекса вполне оправдан.
Если Вы не хотите задумываться над выбором начального значения индекса и, работая с системой, где это значение равно 0, используете только значения индекса от 1 до максимального, то тем самым впустую тратите часть памяти ЭВМ. Например, для массива вещественных чисел, описанного оператором DIM A (100, 100) , требуется по 4 байт (в Бейсике BBC — по 5 байт) на каждый элемент массива, т. е. всего 4*101*101 =40 804 байт при начальном значении индекса, равном 0 (в Бейсике BBC 51 005 байт). Если же работать с индексами от 1 до максимального, то реально используется память объемом в 4 * 100 * 100 = 40 000 байт (в Бейсике BBC 50 000 байт). Таким образом, 804 байт (в Бейсике BBC 1005 байт) никогда не будут использоваться и окажутся недоступными и для других нужд, например для хранения команд программы или строк символов. В этом случае указание в начале программы оператора OPTION BASE 1 сбережет немалый объем памяти.
В некоторых системах предусмотрены специальные функции для работы с матрицами; имена этих функций начинаются с MAT, например MAT INPUT. Все эти функции оперируют элементами массива, индексы которых изменяются от 1 до максимального, независимо от того, существуют ли элементы с нулевыми индексами (см. гл. 7) .
4.1.3. ИНДЕКСЫ МАССИВОВ
Многие системы с Бейсиком позволяют в качестве индекса использовать не только переменные (например, M(D)), но любые арифметические выражения. Так, если значение D равно 3, то
M (D + 2) эквивалентно M (5), M(6*D-12) эквивалентно M(6), M(2*(D-4)+5) эквивалентно M (3) .
Следите за тем, чтобы при вычислении выражений получались нужные целые результаты, так как разные системы поступают с дробной частью индекса по-разному. Некоторые из них отбрасывают дробную часть, так что 7.2, 7.5 и 7.9 будут заменены на 7, а другие округляют значение индекса до ближайшего целого, заменяя дробную часть от 0.5 и больше на 1. В этом случае 7.2 заменяется на 7, а 7.5 и 7.9 — на 8.
Лучше всего либо пользоваться в индексных выражениях целыми переменными (если таковые допускаются системой), либо применять функцию INT и свою собственную систему округления. При значении D, равном 5, Вы можете рассчитывать получить в результате вычисления выражения 2*(D-4)+5 число 7, но в действительности результат может оказаться равным 6.999999, и при отбрасывании дробной части получится 6. Это приведет к неправильному выбору элемента массива, и распознать такую ошибку очень трудно. С другой стороны, при D = 5
INT (2*(D-4) +5+0.5) = INT(6.999999 + 0.5) = INT (7.499999) = 7

что и требовалось. Таким образом, указанные выше элементы массивов надежнее определить как M(INT+2.5)), M(INT(6*D-12+0.5)) и M(INT(2*(D-4)+5.5)).

4.1.4. ЭКОНОМИЧНОЕ ИСПОЛЬЗОВАНИЕ МАССИВОВ

При решении некоторых задач девять из десяти начинающих программистов непременно используют массив, хотя без него можно и обойтись. На первых порах это не вызывает трудностей, потому что написанные такими программистами программы обычно невелики и без труда помещаются в память ЭВМ. Однако если эта практика войдет в привычку, то в программах большего размера будет понапрасну тратиться много памяти. Чтобы исключить неоправданное использование массива, полезно задаться вопросом: потребуются ли значения элементов массива более одного раза? Если нет, то массив, скорее всего, не нужен. Вернемся к программе из подразд. 4.1.1: если бы от нас не требовалось вычислить значения запасов товаров в процентах от суммарного запаса, то программу можно было бы радикальным образом изменить. В исходном варианте программы надо было ввести все значения, подсчитать их сумму, вывести все эти значения, добавив колонку, в которой каждое значение было поделено на эту сумму. Поэтому понадобилось два цикла. Изменяя задачу и не требуя выдачи значений запасов в процентах от общей суммы, приходим к следующей структуре:

которой соответствует программа

```
10 REM ИНВЕНТАРИЗАЦИОННАЯ ОПИСЬ. НЕМЕДЛЕННАЯ ПЕЧАТЬ
20 T=0
30 FOR I=1 TO 20
40   INPUT S
50   T=T+S
60   PRINT "ТОВАР" ; I ; "ЗАПАС" ;S
70 NEXT I
80 PRINT "ВСЕГО";T;"ШТУК"
90 END
```

Ясно, что этим вариантом программы труднее пользоваться, так как изображения вводимых и выводимых значений перемешаны между собой:


```
RUN
?6
ТОВАР 1 ЗАПАС =6
?4
ТОВАР 2 ЗАПАС =4
...
...
```

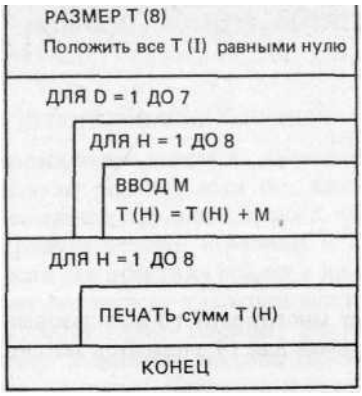
Однако при этом за счет многократного использования одной переменной экономится память, требуемая для 19 элементов массива.

4.1.5. ПРИМЕР ПРОГРАММЫ

Приведенные ниже программы показывают, как можно упростить программирование за счет совместного применения массивов и циклов FOR. Начнем с простой программы, суммирующей количество пинт молока, доставляемых в один дом в течение недели. Структура этой программы по существу та же, что и программы, обсуждавшейся выше в подразд. 4.1.4; сама же программа такова:

```
10 REM ДОСТАВКА МОЛОКА В ОДИН ДОМ
20 T=0
30 FOR D=1 TO 7
40   INPUT M,
50   T=T+M
60 NEXT D
70 PRINT "ВСЕГО ДОСТАВЛЕНО";T
80 END
RUN
?1.3.0.2.1.2.1
ВСЕГО ДОСТАВЛЕНО 10
END AT LINE 80
```

Обратите внимание на запятую, указанную в качестве разделителя в операторе INPUT в строке 40. Она означает, что при каждом исполнении этого оператора перехода на новую строку не происходит, так что требуемые значения можно вводить, указывая их в одной строке и разделяя запятыми. Некоторые системы позволяют вместо запятой указывать в операторе INPUT двоеточие. В Бейсике BBC нет ни той ни другой формы, поэтому каждое вводимое значение придется набирать на новой строке. Теперь обобщим нашу задачу следующим образом: пусть молоко доставляется в восемь домов и в конце недели требуется подсчитать, сколько всего молока было доставлено в каждый дом. Таким образом, простая переменная T в приведенной выше программе становится массивом из восьми элементов, в каждом из которых накапливается количество молока, доставленного в соответствующий дом. Структура этой программы такова:



Первый прямоугольник на этой структограмме в действительности является циклом, служащим для присваивания 0 всем элементам массива T. Но так как это действие логически завершено и не разъясняет процесс работы над задачей, то оно изображено в виде блока. Эту структограмму можно развить до следующей программы:

```
5 REM ДОСТАВКА МОЛОКА В 8 ДОМОВ
10 OPTION BASE 1
20 DIM T(8)
30 FOR I=1 TO 8
40   T(I)=0
50 NEXT I
```

```
60 REM
70 FOR D=1 TO 7
60   PRINT "ДЕНЬ" ;D
90   FOR H=1 TO 6
100     INPUT S,
110     T<H)=T(H)+S
120   NEXT H
130 NEXT D
140 REM
150 FOR H=1 TO 8
160   PRINT "В ДОМ";H;"ДОСТАВЛЕНО";T(H);"ПИНТ МОЛОКА" 170 NEXT H 180 END
RUN ДЕНЬ 1
?1.4.1.3.2.0.5.2 ДЕНЬ 2
?3.0.1.1.1.2.0.1
ДЕНЬ 3
?0.2.0.1.4.0.0.1
ДЕНЬ 4
?2.l.l.0.1.1.2.1
ДЕНЬ 5
?1.1.2.1.0.0.1.0
ДЕНЬ 6
?2.1.0.0.0.3.4.1
ДЕНЬ 7
?1.5.2.8.1.1.3.1
96


В ДОМ 1 ДОСТАВЛЕНО   10 ПИНТ МОЛОКА
В ДОМ 2 ДОСТАВЛЕНО   14 ПИНТ МОЛОКА
В ДОМ 3 ДОСТАВЛЕНО  7 ПИНТ МОЛОКА
В ДОМ 4 ДОСТАВЛЕНО  6 ПИНТ МОЛОКА
В ДОМ 5 ДОСТАВЛЕНО   9 ПИНТ МОЛОКА
В ДОМ 6 ДОСТАВЛЕНО  7 ПИНТ МОЛОКА
В ДОМ 7 ДОСТАВЛЕНО   15 ПИНТ МОЛОКА
В ДОМ 8 ДОСТАВЛЕНО  7 ПИНТ МОЛОКА
END AT LINE 180
```

Продавец молока набирает на клавиатуре числа, показывающие, сколько пинт молока отправлялось каждый день в каждый дом. Для первого дома, с номером 1, данные уже обрабатывались простой программой. Действия же этой программы состоят попросту в сложении всех чисел в каждом столбце и в изображении полученных сумм.

4.1.6. ВВОД И ВЫВОД ДАННЫХ С ПОМОЩЬЮ ОПЕРАТОРА MAT

Оператор MAT предназначен для проведения манипуляций всеми элементами массива, начиная с индекса 1, в одном операторе Бейсика. Как указано в гл. 7, во многих системах не предусмотрено полного спектра таких манипуляций, но, по крайней мере, в нескольких системах предусмотрен ввод и вывод данных с помощью оператора MAT.

Попросту говоря, операторы

```
10   DIM A (5)
20   MAT INPUT A
```

обеспечивают ввод данных в пять элементов массива A (1), A(2), A(3), A (4) и A (5). Вводимые значения можно набирать на одной строке или распределять по нескольким строкам. Если Ваша

система не позволяет пользоваться оператором MAT, то указанные выше операторы можно заменить на следующий эквивалент:

```
10   DIM A(5)
20   FORI=1 TO 5
30     INPUT A(I),
40   NEXT I
```

Запятая после A(I) как раз и позволяет вводить несколько значений в одной строке, разделяя их запятыми. Если такая возможность в Вашей системе отсутствует, Вам придется набирать каждое значение на отдельной строке. Вывод работает точно так же: операторы

```
10   DIM P(8)
20   MAT PRINT P
```

выведут восемь значений P(1) — P(8), каждое на новой строке. Если после P указать запятую или точку с запятой, то значения будут выводиться в одной строке либо с большими интервалами, либо вплотную. Если Ваша система не позволяет пользоваться оператором MAT, то предыдущие операторы можно заменить на следующий эквивалент:

```
10   DIM P(8)
```



```
20   FOR I = 1 TO 8 30      PRINT P (I) 40   NEXT I
```

Как MAT INPUT, так и MAT PRINT оперируют и двумерными массивами. При работе с оператором INPUT тщательно установите, в каком порядке значения присваиваются отдельным элементам массива; обычно первым изменяется второй индекс. Таким образом, операторы

```
10   DIM A (2,3) 20   MAT INPUT A
```

эквивалентны операторам

```
10   DIM A (2,3)
20   FOR I = 1 TO 2
30     FOR J = 1 TO 3
40    INPUT A (I, J),
50     NEXT J
60   NEXT I
```

Вывод осуществляется аналогичным образом: первым изменяется второй индекс. Например,

```
10  REM ФРАГМЕНТ ДЛЯ ДЕМОНСТРАЦИИ ОПЕРАТОРА MAT
20  DIM A(2,3)
30   MAT INPUT A
40   REM
50  MAT PRINT A,
60   END
RUN
?1.2.3.4.5.6
1                    2                    3
4                    5                    6
END AT LINE 60
```

Матрица A размерами 2X3 изображается в обычной алгебраической форме в виде строк и столбцов, а запятая в операторе MAT PRINT вызывает, как показано раньше, вывод элементов матрицы по строкам.

В Бейсике BBC эти операторы отсутствуют, но, в отличие от большинства других систем, в этой версии Бейсика есть возможность выделения "процедур", которыми можно воспользоваться для определения процедур MATINPUT(A) и MATPRINT(A), выполняющих описанные выше действия.

4.2. СТРОКИ СИМВОЛОВ

Программа на языке Бейсик может без труда манипулировать текстом, т. е. строками символов. Будучи не слишком элегантны вследствие того, что обеспечиваются библиотекой встроенных функций, эти манипуляции достаточно эффективны.

Поэтому строки символов могут рассматриваться как другой тип объектов, которыми можно манипулировать средствами Бейсика. В некото-

рых языках программирования делаются попытки как можно шире распространить этот вид манипуляций на все возможные объекты, но немногие языки преуспели в этом. В языке Бейсик и попытки не делается! Каждый тип объектов и ассоциированных с ним операций обычно отделен от других типов и операций, поэтому чисто внешне Бейсик представляет собой ряд отдельных, слабо связанных частей. Правда, Бейсик имеет то достоинство, что не только теоретически способен достаточно легко расширяться и изменяться, но и постоянно делает это.

4.2.1. СТРОКОВЫЕ ПЕРЕМЕННЫЕ

Строка рассматривается как группа символов. В программе строки появляются в виде заключенных в кавычки групп символов и особенно интенсивно употребляются в операторах PRINT, например:

```
10   PRINT "ДОМ";Н;Т(Н) ; "ПИНТ МОЛОКА ОТПРАВЛЕНО"
```

Строка может содержать любую букву алфавита, любую цифру и любой из специальных символов, таких как "/£?;&=$@+-#., и пробел. Она может содержать и не печатаемые символы, вызывающие при печати строки некоторые действия, например перемещение курсора. В отличие от большинства операторов Бейсика, в строке пробел учитывается.

До сих пор мы в основном имели дело с переменными, способными хранить только числовые значения. Однако существует и другой тип переменных, которые способны хранить только строки и называются строковыми. Имена этих переменных выбираются по общему правилу, но в конце имени добавляется знак $. Между одноименными числовыми и строковыми переменными не должно быть никакой связи; поэтому строковая переменная A$ должна полностью отличаться от числовой переменной A. Ниже приводится мини-программа, иллюстрирующая применение строковой переменной:

```
10 REM ПРИМЕР РАБОТЫ СО СТРОКАМИ СИМВОЛОВ
20 PRINT "НАЗОВИТЕ ВАШУ ФАМИЛИЮ. ПОЖАЛУЙСТА ";
30 INPUT A$
40 PRINT "ДОБРЫЙ ДЕНЬ, ";A$;"! КАК ПОЖИВАЕТЕ?"
50 END
RUN
НАЗОВИТЕ ВАШУ ФАМИЛИЮ, ПОЖАЛУЙСТА
?УОЛШ
ДОБРЫЙ ДЕНЬ, УОЛШ! КАК ПОЖИВАЕТЕ? END AT LINE 50
```

Запустим ее еще раз: RUN

```
НАЗОВИТЕ ВАШУ ФАМИЛИЮ, ПОЖАЛУЙСТА
?27.6! ДОБРЫЙ ДЕНЬ,27.6!
КАК ПОЖИВАЕТЕ? END AT LINE 50
```

Все, что, дается в качестве ввода, помещается в строковую переменную и позже распечатывается. Единственное ограничение состоит в том, что в

Бейсике при вводе оператором INPUT запятая воспринимается как разделитель элементов данных. Повторный запуск программы

```
RUN
НАЗОВИТЕ ВАШУ ФАМИЛИЮ, ПОЖАЛУЙСТА
?УОЛШ, ЛАРА
WARNING - EXTRA INPUT IGNORED (Предупреждение - дополнительный
ввод проигнорирован)
ДОБРЫЙ ДЕНЬ, УОЛШ! КАК ПОЖИВАЕТЕ?
END AT LINE 50
```

показывает, что запятая и указанные после нее символы игнорируются. Если бы в строке 30 было указано INPUT A$, B$, то первый из приведенных выше элементов данных (УОЛШ) был бы помещен в A$, а второй (ЛАРА) -в B$. Правда, есть средство считать всю строку одной переменной невзирая на число запятых в строке (см. в подразд. 4.2.4) .

Другой способ считать строку сложного вида в одну переменную, - заключить ее в кавычки. Повторим запуск программы:

```
RUN
НАЗОВИТЕ ВАШУ ФАМИЛИЮ, ПОЖАЛУЙСТА? "УОЛШ,ЛАРА"
ДОБРЫЙ ДЕНЬ, УОЛШ,ЛАРА! КАК ПОЖИВАЕТЕ?
END OF LINE 50
```

Как и числовым строковым переменным в программе можно присваивать значения. Например, допустимы следующие операторы:

```
10   A$="ЛАРА"
10  C$= '£25.6"
10   B$=C$
10   IF Y$ = "ДА" THEN 200
10  IF X$ <> AS THEN PRINT "ВСЕ В ПОРЯДКЕ"
```

Приведенные выше операторы IF проверяют совпадение двух строк. Пробелы принимаются во внимание. Если бы Y$ содержала " ДА" (обратите внимание на ведущий пробел), то Y$ не совпала бы с "ДА".

Строки могут содержать от 0 до очень большого числа символов. Нулевое число символов содержит строка "", называемая пустой. Максимально допустимая длина строки варьируется от системы к системе (табл. 4.1) .

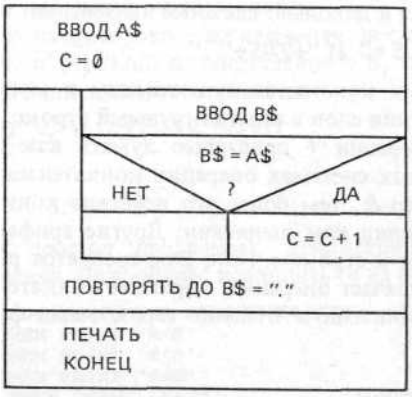
Таблица 4.1.*Максимальная длина строки в различных системах*

Система	BBC	Microsoft	CBM PET	Sinclair	ICL 2903/4
Максимальная длина строки (в символах)	255	255	255	Без ограничений	511

Могут существовать и ограничения на число символов, которые можно поместить в строку за один прием при вводе с клавиатуры или при составлении оператора программы. Обычно эти ограничения связаны с реализацией системы и определенным образом зависят от длины строки экрана ВТУ. В типичных случаях допустимый максимум равен длине строки (скажем, 40 или 60 символов), а более длинные строки могут быть образованы слиянием нескольких строк.

Приведенная ниже программа довольно элементарна и предназначена для подсчета слов. Распознаваемое слово вводится первым; затем вводится предложение, завершаемое точкой. Каждая

составная часть этого предложения набирается на новой строке. Структограмма этой программы такова:



Обратите внимание на использованную выше диаграмму "ПОВТОРЯТЬ ДО". Она указывает на то, что проверка происходит в конце цикла. В большинстве систем такую конструкцию можно реализовать с помощью оператора IF, но в некоторых системах с Бейсиком для этой цели предусмотрен оператор REPEAT-UNTIL (см. гл. 6). Сама программа такова:

```
10 REM ЭЛЕМЕНТАРНЫЙ СЧЕТЧИК СЛОВ
20 INPUT A$
30 C=0
40   INPUT B$
50   IF B$=A$ THEN C=C+1
60 IF B$<>"." THEN 40
70 REM
80 PRINT "СЛОВО ";A$;" ВСТРЕТИЛОСЬ";C;"РАЗ"
90 END

RUN
?ДВА          <------------ вводится в A$
?ДВА          <------------
?И                        !
?ДВА                      !- вводятся в B$
?РАВНО                    !
?ЧЕТЫРЕ                   !
?.            <------------
СЛОВО ДВА ВСТРЕТИЛОСЬ 2 РАЗА
END AT LINE 90
```


4.2.2. СЛИЯНИЕ СТРОК

Слияние строк формирует более длинные строки и похоже на составление поезда из вагонов. Это действие называется конкатенацией. Для конкатенации двух строк в одних системах между строками указывается знак &, в других — знак +, например:

```
10 A$="УОЛТ"
20 B$=A$+" ДИСНЕЙ"
30 PRINT B$
40 END
RUN
УОЛТ ДИСНЕЙ
END AT LINE 40
```

Могут образовываться и довольно сложные выражения:

```
10   A$=C$+" "+D$+" Н"+Z8$+"."
```

Пробелы в строковых константах существенны и могут применяться для предотвращения слияния слов в результирующей строке.

Поскольку об операции + привычнее думать как об арифметическом сложении, в некоторых системах операция конкатенации обозначается другим знаком, например &, тем более что действие конкатенации состоит не в сложении, а в слиянии или сцеплении. Другие арифметические операции

*, /, - не применимы к строкам. Если Вам требуется разбить строку на отдельные части, что отвечает операции, обратной конкатенации, то надо пользоваться предусмотренными в Бейсике строковыми функциями (см. подразд. 4.2.6).

4.2.3. СРАВНЕНИЕ СТРОК

Строки можно указывать в условиях оператора IF в сочетании со всеми операциями сравнения. При проверке на совпадение строки по обеим частям от знака = должны быть идентичны с учетом как ведущих, так и концевых пробелов. Например, условие оператора

IF A$="ПОЧЕМУ" THEN 100

выполнено в том случае, если значение A$ равно "ПОЧЕМУ", но не выполнено, если оно равно " ПОЧЕМУ", или "ПОЧЕМУ ", или любому другому

значению.

Можно использовать и операции < (меньше) или > (больше). Применительно к строкам операция < интерпретируется как "младше в алфавитном порядке", а операция > - как "старше в алфавитном порядке".

Только что сказанное требует некоторых оговорок, а именно как поступать со специальными символами или цифрами? Ответ на этот вопрос прост: при сравнении строковых значений ЭВМ сопоставляет внутренние числовые коды символов и на основе этого сопоставления устанавливает выполнение отношения < или >. Так, если "символ 1" имеет меньший числовой код, чем "символ 2", то "символ 1" < "символ 2". В большинстве ЭВМ (но не во всех) для внутреннего представления символов используются коды ASCII (см. приложение II). В любом случае в Вашей системе должна быть предус-

Таблица 4.2. *Некоторые примеры кодов* ASCII

Символ	*	+	-	0	9	A	H	Z	a	h	z
Код (деся-тичный)	42	43	45	48	57	65	72	90	97	104	122

мотрена функция, позволяющая найти числовой код, соответствующий каждому символу. В табл. 4.2 приведено несколько примеров кодов ASCII. Учтите, что некоторые специальные символы предшествуют цифрам 0 — 9, а сами цифры предшествуют прописным буквам. Строчные буквы оказываются почти в самом конце списка символов[1].

Строки нетрудно отсортировать по алфавиту, поскольку между строкой, начинающейся с A, и строкой, начинающейся с B, справедливо отношение <, и т. д. Например, выполнены соотношения

"CAR" < "CARD" < "CARTER" < "CAT" а также

Ниже приводится простая программа, демонстрирующая некоторые из подобных соотношений. Попробуйте исполнить ее на своей ЭВМ.

```
10 REM ТЕСТ, ДЕМОНСТРИРУЮЩИЙ СРАВНЕНИЕ СТРОК СИМВОЛОВ
20 IF IF "A"<"B" THEN PRINT "A<B"
30 IF "0"<"8" THEN PRINT "0<8"
4(8 IF "0"<"D" THEN PRINT "0<D"
50 IF "X" = "XX" THEN PRINT "X=XX"
60 IF "X"<"XX" THEN PRINT "X<XX"
70 IF "CART HORSE"<"CARTER" THEN PRINT "CART HORSE<CARTER"
80 END
RUN
A<B
0<8
```

```
0<D
X<XX
CART HORSE<CARTER
END AT LINE 80
```

Следующая программа иллюстрирует элементарный метод сортировки трех строк по алфавиту. Если бы вместо строковых были использованы числовые переменные, то программа отсортировала бы их в порядке возрастания. Для сортировки большого числа строк использованный в этой программе метод

Все сказанное относится к латинским буквам. Во многих отечественных вычислительных системах и внешних устройствах принята система кодирования, в которой прописные русские буквы замещают строчные латинские и еще несколько относительно редко используемых знаков. Таким образом получается набор символов, включающий прописные буквы латинского и русского алфавитов и не содержащий строчных букв. К сожалению, в этой системе коды прописных русских букв не упорядочены по алфавиту и отсортировать строки из русских букв так, как показано далее, нельзя. — *Прим. перев.*

малопригоден. Гораздо более эффективная программа сортировки приводится в разд. 4.5.

```
10   REM ЭЛЕМЕНТАРНАЯ СОРТИРОВКА
28    INPUT A$,B$,C$
30   REM ПРОВЕРИМ ПЕРВУЮ ПАРУ И ПОМЕНЯЕМ МЕСТАМИ, ЕСЛИ
35   REM   ИДУТ НЕ ПО ПОРЯДКУ
40   IF A$<B$ THEN 90 50   X$=A$
60   A$=B$
70   B$=X$
80  REM ПРОВЕРИМ СЛЕДУЮЩУЮ ПАРУ И ПОМЕНЯЕМ МЕСТАМИ, ЕСЛИ
85  REM  ИДУТ НЕ ПО ПОРЯДКУ
90   IF B$<C$ THEN 150 100   X$=B$
110  B$=C$
120   C$=X$
130  REM ТАК КАК ЗНАЧЕНИЯ ПЕРЕСТАВЛЕНЫ. ВНОВЬ ПРОВЕРИМ
135   REM   ПЕРВУЮ ПАРУ
140   GOTO 40
150  PRINT "АЛФАВИТНЫЙ ПОРЯДОК СЛОВ:"
160  PRINT AS,B$,C$
170   END
RUN
?LARA,ANN,BETTY
АЛФАВИТНЫЙ ПОРЯДОК СЛОВ:
ANN                 BETTY               LARA
END AT LINE 170
```

4.2.4. ВВОД СТРОКИ СИМВОЛОВ

Как уже обсуждалось в подразд. 4.2.1, при вводе значения строковой переменной оператором INPUT возникают определенные трудности, если вводимая строка содержит запятые. Запятая служит разделителем, заставляющим оператор INPUT рассматривать такую строку как несколько строк. Заключив строку в кавычки, можно заставить оператор INPUT воспринимать ее как одну строку, но это неудобно, а иногда даже и невозможно, например, если ввод осуществляется оператором INPUT из файла, а не с терминала. Чтобы обойти эту проблему, во многих системах с Бейсиком предусмотрена модификация оператора INPUT, предназначенная специально для ввода в одну

строковую переменную следующей полной строки текста независимо от ее содержания. В табл. 4.3 приводится перечень имен такой модификации для различных систем. Действие каждого типа модификации то же, что и у обычного оператора INPUT в этой системе. Например, можно указывать приглашение к вводу данных, если это допускается в операторе INPUT.

Таблица 4.3. *Операторы ввода строки для различных систем*

Система	BBC	Microsoft	ICL 2903/4
Форма записи оператора ввода строки	INPUT LINE A$,B$	LINE INPUT A$, B$	LINPUT A$, B$

О работе этих операторов можно судить по действию следующей программы для Бейсика Microsoft:

```
10 REM ВВОД СТРОКИ
20 LINE INPUT A$,B$
30 PRINT
40 PRINT "ПЕРВАЯ СТРОКА: ";A$
50 PRINT "ВТОРАЯ СТРОКА: ";B$
60 END
RUN
НЕСКОЛЬКО СИМВОЛОВ ДЛЯ ПРИМЕРА, " И !&
ЕЩЕ НЕСКОЛЬКО СИМВОЛОВ НА СЛЕДУЮЩЕЙ СТРОКЕ. $"":
ПЕРВАЯ СТРОКА: НЕСКОЛЬКО СИМВОЛОВ ДЛЯ ПРИМЕРА, " И !&
ВТОРАЯ СТРОКА: ЕЩЕ НЕСКОЛЬКО СИМВОЛОВ НА СЛЕДУЮЩЕЙ СТРОКЕ, $"";
END AT LINE 60
```

4.2.5. ЧИСЛОВЫЕ ФУНКЦИИ СО СТРОКОВЫМИ АРГУМЕНТАМИ

Все функции Бейсика для работы со строками машинно-зависимы. Ниже описываются наиболее широко распространенные функции.

В данном подразделе речь пойдет о функциях, которые обрабатывают строки символов, но возвращают числовое значение, например длину строки. Будучи функциями, они требуют один или несколько аргументов и возвращают одно числовое значение, присваиваемое имени функции. Например, оператор

```
10   A =LEN("DON'T GIVE UP")
```

поместит в переменную A значение 13 в результате обращения к функции LEN, возвращающей длину строки. В табл. 4.4 - 4.8 приводятся альтернативные имена и указываются отдельные особенности реализаций для наиболее распространенных функций. Для единообразия действие этих функций иллюстрируется на примере версии Бейсика Microsoft. В этих таблицах A$ и B$ могут быть переменными, выражениями или константами.

Если A$ содержит строку "ФОСФОР", то в результате выполнения операторов

```
B=LEN(A$)
C = ASC (A$)
D=INSTR(A$,""Ф")
E=INSTR(3,A$,"ФО")
```

Таблица 4.4 *Функции Бейсика для персональной ЭВМ* ZX81 *фирмы* Sinclair

Результат	Функция	Действие
Значение кода	CODE A$	Дает внутренний машинный код первого символа строки A$. Если строка пуста, в качестве результата возвращается 0

Длина строки	LEN(A$)	Дает число символов в строке A$
Число	VAL(A$)	Дает численный результат обработки строки A$ которая может содержать символьную запись достаточно сложного арифметического выражения

Таблица 4.5. *Функции Бейсика* Microsoft; *некоторые другие функции используются при работе с файлами прямого доступа (см. подразд. 8.6.2)*

Результат	Функция	Действие
Значение кода	ASC(A$)	Дает (внутренний машинный) коя первого символа
Длина строки	LEN(A$)	Дает число символов в строке A$
Позиция	INSTR(A$,B$) INSTR(I,A$,B$)	Дает начальную позицию первого вхождения строки B$ в строку A$; возвращает 0, если строка B$ не является вырезкой из строки A$. Если параметр I указан, то поиск вхождения строки B$ в строку AS начнется с 1-й позиции строки A$
Число	VAL(A$)	Дает число по его символьной записи в строке AS которая может содержать только цифры или знаки +, -, или &. Пробелы и символы прогона строки при этом игнорируются

Таблица 4.6. *Функции Бейсика* BBC

Результат	Функция	Действие
Значение кода	ASC(A$)	Дает (внутренний машинный) код ASCII первого символа строки A$ (т. е. положительное число или 0) . Если строка пуста, то возвращается значение -1
Длина строки	LEN(A$)	Дает число символов в строке A$
Позиция	INSTR(A$,B$) INSTR(A$,B$,I)	Дает начальную позицию первого вхождения строки B$ в строку A$; возвращает 0, если строка B$ не является вырезкой из строки AS. Если параметр 1 указан, то поиск вхождения строки B$ в строку AS начнется с 1-й позиции строки A$
Число	VAL(A$)	Дает число по его символьной записи в строке AS; если в строке содержатся посторонние символы, то возвращается 0

Таблица 4.7. *Функции Бейсика для персональной ЭВМ* CBM PET

Результат	Функция	Действие

Таблица 4.8. *Функции Бейсика для ЭВМ* ICL *2903/4*

Результат	Функция	Действие
Значение кода	CHR(A$)	Дает внутренний машинный код первого символа строки A$
Длина строки	LEN(A$)	Дает число символов в строке A$
Количество вхождений	OCC(A$,B$)	Дает число неперекрывающихся вхождений строки B$ в строку A$
Позиция	POS(A$,B$) POS(A$,B$,N)	Дает начальную позицию первого вхождения строки B$ в строку A$; возвращает 0, если строка B$ не является вырезкой из строки A$. Если параметр N указан, то поиск вхождения строки B$ в строку A$ начнется с N-й позиции строки A$
Число	VAL(A$)	Дает число по его символьной записи в строке A$; строка не должна содержать посторонних символов

В получит значение 6, С 102, равное десятичному коду буквы Ф, D1, указывающее позицию символа в строке A$, начиная с которой произошло совпадение с Ф, а Е получит значение 4, указывающее позицию символа в строке A$, начиная с которой произошло совпадение с ФО при поиске от позиции 3 к концу строки. Использование VAL(A$) ошибочно, и для данной системы приведет к значению 0.

Если A$ содержит "-12356", то VAL(A$) возвратит число -12356, которое при необходимости можно обрабатывать арифметически. Существует функция, обратная VAL и преобразующая числовое значение в строку, содержащую запись этого числа в определенной системе счисления. Такую строку полезно представлять себе как изображение числа. Если A$ содержит "1234", а B$ "5678", то в результате исполнения операторов

C$=A$ + B$ E=VAL(A$) +VAL(B$)

C$ получит значение "12345678", а Е 6912. Функция VAL может показаться странной, но она оказывается чрезвычайно полезной в двух случаях. Во-первых, ее можно применять для придания выводу требуемого формата. Можно преобразовывать подлежащие выводу числа в строки, манипулировать этими строками, добавляя или убавляя пробелы, отбрасывая ненужные цифры, затем выдавать результаты оператором PRINT, сразу или предварительно объединяя несколько строк в одну, и т. д.

Во-вторых, при некоторых видах обработки дисковых файлов требуется, чтобы обмен с диском происходил только строками символов. В этих случаях приходится преобразовывать при записи на диск числовые значения в строки символов, а при чтении с диска — из строк символов обратно в числовые значения.

Весьма полезной оказывается функция OCC (см. табл. 4.8), и если она потребовалась, а ее не оказалось в Вашей системе, то приведем фрагмент

программы, выполняющий ту же работу. В нем A$ воспринимается как аргумент, а B$ должна содержать искомый образец. Возвращаемый в переменной О результат равен 0, либо числу обнаруженных в A$ неперекрывающихся вхождений образца. Обратите внимание на то, что функция INSTR применяется так, как описано в табл. 4.5. Для решения этой задачи, иллюстрируемого приведенной ниже структограммой, нам требуется продолжать сравнение образца B$ с символами

строки A$ до тех пор, пока не произойдет одно из двух событий: либо совпадений с образцом больше нет, либо при поиске мы вышли за пределы строки A$. Так как структурного символа, позволившего бы осуществить выход из различных мест внутри цикла, не существует, то условие выхода из цикла надо сформировать в его конце. На практике для этого вводится дополнительная переменная M. Ей присваивается начальное значение 0, а затем в случае выполнения какого-либо из условий значение 1. При этом цикл управляется по принципу "повторять, пока не будет достигнут конец строки A$ либо пока не станет ни одного совпадения с образцом", что в терминах переменной M примет вид "повторять, пока M не станет равной 1".

Ниже приводится соответствующая этой структограмме программа. Она не представляет собой полную программу, так как предназначена для использования в виде части большой программы — например, в виде подпрограммы или функции (эти понятия обсуждаются в гл. 5).

```
100 REM ФРАГМЕНТ ПРОГРАММЫ ДЛЯ ОПРЕДЕЛЕНИЯ ЧИСЛА ВХОЖДЕНИЙ
110 0=8
120 1=1
130 J=LEN(A$)
108



140   K=LEN(B$)
150   M=0
160   P=INSTR (I,A$,B$)
170   IF P<>0 THEN 200
180  M=1
190     GOTO 230
200   O=O+1
210   I=P+K
220   IF (I+K)>J THEN M=l
230 IF M<>1 THEN 160
240 REM ВЫХОД С ИСКОМЫМ ЗНАЧЕНИЕМ В O
```

Для проверки этого фрагмента добавим к нему строки с операторами INPUT, PRINT и END:

```
10 INPUT A$.B$
250 PRINT "ЧИСЛО ВХОЖДЕНИЙ =";O
```

```
260 END
RUN
?ВОТ КОТ УВИДЕЛ БЕГЕМОТА.ОТ
ЧИСЛО ВХОЖДЕНИЙ = 3
END AT LINE 260
```

Эта программа дополнительно регистрирует позиции в строке A$, начиная с которых в этой строке встречается искомый образец. Позиции можно запомнить в массиве и использовать по окончанию процесса поиска вместе с числом вхождений образца в строку A$.

4.2.6. ВЫРЕЗКИ И ФУНКЦИИ

В этом подразделе обсуждаются функции, возвращающие в качестве результата строки, являющиеся либо вырезками из уже существующих строк, либо новыми строками, получающимися, например, в процессе преобразования числового значения в строку символов. Здесь описываются только широко распространенные функции, а для примеров, если не оговорено, используются функции Бейсика Microsoft. Перечни различных функций, сгруппированных по характеру их действия, приводятся в табл. 4.9 — 4.13.

Основное внимание уделяется тем функциям, которые возвращают части строк или, более точно, вырезки. Вырезка представляет собой последовательность смежных символов, извлеченную из другой строки. Так, "ДНЕМ" и "М РОЖДЕНИЯ" представляют собой вырезки из строки "С ДНЕМ РОЖДЕНИЯ", а "СДЕНИЯ" и "НЕРОЖ" - нет, хотя и являются новыми строками, которые можно получить из нескольких вырезок исходной строки.

Если строка A$ содержала "ДОБРЫЙ ДЕНЬ", то в результате выполнения операторов

```
A1$ = LEFT$(A$,9) A2$ = LEFT$(A$,2) B1$ = RIGHT$(A$,4) B2$ = RIGHT$(A$,9)
```


Таблица 4.9. *Функции Бейсика со строковыми значениями для персональной ЭВМ* ZX81 *фирмы* Sinclair

Результат	Функция	Действие
Преобразование кода в символ	CHR$ N	Дает символ, код которого равен N и находится в пределах 0 ... 255
Вырезки строки	Выделяется часть строки; не является функцией	Выражение в скобках определяет часть строки от N-го до M-го символов включительно. Если N не указано, оно полагается равным 1. Если не указано M, оно полагается равным числу символов в строке
	Строка (NTOM)	
	Строка (N)	В такой форме дает ровно один N-й символ строки
Символьная запись числа	STR$N	Дает строку, являющуюся символьной записью числа N. Обратной функцией является VAL(более общая функция, так как может входить в состав выражений)
Особый ввод символа	INKEY$	Не имеет аргументов; обеспечивает ввод с клавиатуры одного символа

Таблица 4.10. *Функции Бейсика* Microsoft *со строковыми значениями*

Результат	Функция	Действие
Преобразование кода в символ	CHR$(N)	Дает символ с кодом ASCII, равным N

Вырезки строки	LEFT$(A$,N) RIGHT$(A$,N) MID$(A$,N,M)	Дает N левых (LEFT$) или правых (RIGHT$) символов строки A$ Дает M символов строки A$, начиная с позиции N
Символьная запись числа	STR$(A)	Дает строку, являющуюся символьной записью числа A. Обратной функцией является VAL
Особый ввод символов	INKEY$ INPUT$(N)	Не имеет аргументов и обеспечивает ввод с клавиатуры одного символа Обеспечивает ввод с клавиатуры N символов. (Может быть использована и при работе с файлами)
Средства форматирования вывода	SPACE$(N) STRING$(N,M) STRING$(N,A$)	Дает строку из N пробелов Дает строку из N одинаковых символов с кодом ASCII, равным M, или N-кратно повторяет первый символ строки A$
Разное	HEX$(A) OCT$(A)	Дает строку, содержащую шестнадцатеричную (HEX$) или восьмеричную (OCT$) запись числа A, предварительно округленного до целого значения

Таблица 4.11. *Функции Бейсика* BBC *со строковыми значениями*

Результат	Функция	Действие
Преобразование кода в символ	CHR$(N)	Дает символ, внутренний код которого равен остатку от деления N на 256
Вырезки строки	LEFT$(A$,N) RIGHT$(A$,N) MID$(A$,N,M)	Дает N левых (LEFT$) или правых (RIGHT$) символов строки A$ Дает M символов строки A$, начиная с позиции N
Символьная запись числа	STR$(M)	Дает строку, являющуюся символьной записью числа M. Обратной функцией является VAL
Особый ввод символа	GETS INKEY$(N)	Не имеет аргумента; ожидает ввода следующего символа, который и предоставляет в качестве значения функции Аналогична GETS, но ждет ввода в течение не более чем N отсчетов внутренних часов. Если за это время символ не был введен, то возвращает в качестве значения пустую строку (у этих функций есть версии с числовыми значениями)
Средства форматирования вывода	STRINGS(N,A$)	Дает строку, составленную из N копий строки A$

Таблица 4.12. *Функции Бейсика со строковыми значениями для персональной ЭВМ* CBM PET

Результат	Функция	Действие
Преобразование кода в символ	CHR$(N)	Дает символ с кодом ASCII, равным N

Вырезки строки	LEFT$(A$,N) RIGHT$(A$,N) MID$(A$,N,M)	Дает N левых (LEFT$) или правых (RIGHT$) символов строки A$ Дает M символов строки A$, начиная с позиции N
Символьная запись числа	STR$(A)	Дает строку, являющуюся символьной записью числа A. Обратной функцией является VAL
Особый ввод символа	Не функция, а обычный оператор, напри-мер 10 GET A$	Обеспечивает ввод с клавиатуры в строковую переменную следующего символа

Таблица 4.13. *Функции Бейсика со строковыми значениями для ЭВМ* ICL *2903/4*

Результат	Функция	Действие
Преобразование кода в символ	CHR$(N)	Дает символ, внутренний код которого равен остатку от деления N на 64
Вырезки строки	SEG$(A$,N, M) SUB$(A$,N, M)	в позициях с N-й по M-ю, а в случае SUB$ — из M символов, начиная с N-й позиции строки A$
Символьная запись числа	STR$(A)	Дает строку, являющуюся символьной записью числа A. Обратной функцией является VAL
Средства форматирования вывода	GAP$(N) SGN$(X) LIN$(N) DEL$(A$,B$, N) SDL$(A$,B$, N) REP$(A$,B$, C$,N) SRP$(A$,B$, C$,N)	Возвращает символы +, — или пробел соответственно при положительном, отрицательном или нулевом значении X Дает N символов перехода к новой строке Удаление из строки A$ первого (DEL$) или всех (SDL$) вхождений в нее строки B$ Замена в строке A$ первого (REPS) или всех (SRP$) вхождений в нее строки B$ на строку C$

A1$ будет содержать "ДОБРЫЙ ДЕ", A2$ - "ДО", B1$ - "ДЕНЬ", а B2$ -"БРЫЙ ДЕНЬ". Если нам требуется выполнить присваивания

C1$ = MID$(A$,8,4)
C2$ = MID$(A$,4,8)
C3$ = MID$(A$,4)

то C1$ примет значение "ДЕНЬ", C2$ - "РЫЙ ДЕНЬ", а C3$ получит в качестве значения все символы справа от четвертого, т. е. тоже "РЫЙ ДЕНЬ".

Обычно большинство задач по обработке строк можно решить, комбинируя вырезки из строк с помощью конкатенации. Рассмотрим, к примеру, следующую программу.

```
10 REM ПРИМЕР ДЕЙСТВИЙ НАД СТРОКАМИ СИМВОЛОВ
20 A$="ОКТЯ ДЕКА"
30 B$="БРЬ"
40 FOR I=1 TO 9 STEP 5
50   C$=MIDS$(A$,1,4)
60   C$=C$+B$
70   PRINT C$
```

```
80 NEXT I
90 END
RUN
ОКТЯБРЬ
ДЕКАБРЬ
END AT LINE 90
```


Управляющая переменная цикла по очереди пробегает значения начальных позиций первых частей названий месяцев; в строке 50 первая часть очередного названия месяца вырезается, в строке 60 к ней добавляется "БРЬ", и полученный результат выводится оператором PRINT в строке 70. При желании эти три строки можно объединить в один оператор PRINT MID$(A$, 1,3) + B$.

Если Вы собираетесь приступить к обработке текста или редактированию текстового файла, то стандартные строковые функции могут оказаться не очень удобными, так как при обращении к ним надо указывать номера позиций. Точная форма программных компонент зависит от таких деталей, как тип изменяемых данных и их местонахождение (память или диск); поэтому мы сделаем определенные допущения. Предположим, что данные обрабатываются построчно и текущая строка запоминается в A$. Предположим также, что система должна воспринимать команды изменения текущей строки или замены ее на другую строку.

(а) В удобной для работы системе от пользователя не должно требоваться вычисления номеров позиций для выполнения каких-либо действий над частью строки. Поэтому первой необходимой операцией является ПОИСК B$ в A$, где A$ — текущая строка, а B$ — некоторая вырезка. Эта операция вполне удовлетворительно выполняется функцией INSTR или ей подобной.

(б) Следующая необходимая операция - ВСТАВКА C$ В ПОЗИЦИИ I в A$, что обычно означает вставку новой строки непосредственно перед 1-й позицией исходной строки. Эту операцию можно осуществить с помощью выражения

LEFT$(A$, I-1) + C$ + RIGHT$(A$,LEN(A$)-I+1)

(в) Очередная операция - УДАЛЕНИЕ B$ ИЗ A$. Для ее выполнения сначала можно найти начальную позицию вырезки B$ в A$ так, как указано в (а); если она равна I, то изменение строки можно осуществить с помощью выражения

LEFT$(A$,I-1) + RIGHT$(A$,LEN(A$)-LEN(B$)-I + 1)

(г) Нередко действия (а) , (б) и (в) приходится объединять в одну операцию - ЗАМЕНА B$ НАС$ в A$. Сначала надо определить начальную позицию I вырезки B$ в A$, затем удалить B$ и в заключение вставить C$.

В некоторых системах, например ICL 2903/4 (см. табл. 4.13) предусмотрены функции для выполнения подобных операций, а в других, например BBC, для выполнения этих операций можно создать специальные процедуры, не похожие на функции. Однако во всех системах для выполнения этих операций можно составить подпрограммы.

Образование вырезок удобно и элегантно выполняется системой ZX81 фирмы Sinclair. В скобках после имени переменной указывается диапазон извлекаемых позиций. Например, в результате исполнения операторов

A$ = "OKAY" B$ = A$(1 TO 2)

в B$ окажется "ОК". Кроме того, вырезки можно указать слева от знака присваивания (=) для того, чтобы заменить на новую не всю строку, а только определенную ее часть. Жаль, что большинство систем с Бейсиком предпочитает ограничиваться неуклюжей системой функций.
Вернемся к рассмотрению некоторых из оставшихся строковых функций. Действие функции CHR$ иллюстрируется следующей программой:

```
10 REM ПРИМЕР ДЕЙСТВИЯ ФУНКЦИИ CHR$
20 FOR I=45 TO 69
30   PRINT CHR$(I);
40 NEXT I
50 END
RUN
-. /0123456789: ; = ?@ABCDE
END AT LINE 50
```

Функцией CHR$ можно генерировать символы любого типа, даже не печатаемые. Например, число 13 является десятичным кодом символа возврата каретки, поэтому PRINT CHR$(13) приведет к возврату курсора на начало строки. С помощью CHR$ можно осуществить все типы управления печатающим устройством (например, прогон на начало следующей страницы или многопроходную печать), что никакими другими средствами выполнить не удается. Если Вы попробуете исполнить указанную выше программу для I, меняющегося от 0 до 255, что отвечает полному набору символов ASCII, то увидите очень странные результаты, так как дисплей реагирует на выдаваемые управляющие символы. При этом может оказаться, что работа Бейсика прекратилась и вновь вызвана операционная система!
Функции STR$ и VAL взаимно обратны. Функция VAL очень полезна для извлечения из строк чисел для последующего проведения вычислений. Например, в результате исполнения оператора

```
B = VAL(RIGHT$("ЦЕНА РАВНА 105.12" ,6))
```

в B окажется число 105.12. А функцию STR$ можно использовать для обратного преобразования числа в строку символов. Если B содержит 105.12 и добавляется налог в 15 %, то мы получим 120.888, но так как сумму надо выражать в целых фунтах стерлингов и пенсах, то перед преобразованием полученной суммы в строку символов последнюю цифру 8 надо отбросить. Для этого можно использовать операторы

```
100   B = 105.12
110   B=B* 1.15
120  B$ = STR$(B)
130   B$ = LEFT$(B,6)
140   A$ = "ЦЕНА РАВНА" + B$ +" (ВКЛЮЧАЯ НАЛОГ) " Заметим, что вместо строк 110 - 140 можно написать один оператор
A$ = "ЦЕНА РАВНА" + LEFT$(STR$(B*1.15),6) +"(ВКЛЮЧАЯ НАЛОГ)"
```

В заключение укажем, что во многих случаях бывает полезным иметь возможность ввода одного символа. При обычном вводе программа приоста-

навливается, после чего можно набрать один или несколько символов, но эти символы попадут в программу только после нажатия клавиши возврата каретки. А вот упомянутый выше ввод одного символа не требует команды передачи данных: программа приостанавливается на несколько миллисекунд для ввода и получает либо ранее набранный и хранящийся во входном буфере символ, либо символ, набранный в момент приостанова программы.
Как видно из таблиц, существует много различных форм ввода одного символа; операторы

```
10     A$ = GET$
```

```
10      B$ = INKEY$
10   LET C$ = INKEY$
10  GETD$
```
выполняют одни и те же действия в системах BBC, Microsoft, Sinclair и PET соответственно. В Бейсике PET ввод одного символа реализован как оператор, а не функция и может быть использован с числовой переменной. В Бейсике BBC существует дополнительная функция, ожидающая ввода заданное время. Если за это время ни одного символа не было набрано, то программа продолжает работу и получает в качестве значения пустую строку.

Указанная функция почти всегда используется в цикле для ожидания ввода. Например, при работе с Бейсиком Microsoft цикл
```
50   B$ = INKEY$
60   IF LEN(B$)= 0 THEN 50
70   .....
```
продолжается неопределенное время до тех пор, пока не будет нажата хоть какая-нибудь клавиша. Таким способом можно организовать свою систему ввода, занося набираемые символы в строку до тех пор, пока не будет набран специальный символ или символ возврата каретки (равный CHR$(13)):
```
100 I$=""
110   B$=INKEY$
120   IF LEN(B$)=0 THEN 110
130   IF B$=CHH$(13) THEN 200
140   I$=I$+B$
150  GOTO 110
200  REM В I$ СОДЕРЖИТСЯ ПОСЛЕДОВАТЕЛЬНОСТЬ НАБРАННЫХ СИМВОЛОВ
210   .....
```
Если заменить цикл по значению B$, начатый строкой 110, на цикл FOR-NEXT, обеспечивающий ограниченное число повторений, то тем самым время ожидания ввода будет ограничено. В этом случае программа завершит ввод строки либо после нажатия на клавишу возврата каретки, либо по истечению подходящего промежутка времени.

Так как функции INKEYS (или GETS) дают возможность полного управления вводом, то с их помощью можно конструировать собственные процедуры ввода, например ввода в свободном формате, и т. д.

4.3. МАССИВЫ СТРОК СИМВОЛОВ

В разд. 4.1 мы видели, что группу чисел можно хранить в массиве, скажем, по имени A; при этом числа запоминаются в элементах массива A, а именно A (1), A (2), A (3) и т. д. Аналогично группу строк можно хранить в массиве A$ и обращаться к отдельным строкам по именам A$(1), A$(2) и т. д. Обычно для массивов строк допустимы те же имена, что и для обычных массивов, только к имени добавляется в конце знак $. ,

Наибольшие ограничения на имена массивов, в том числе и массивов строк, которые должны быть однобуквенными, накладываются Бейсиком в системе ICL 2903/4. Другие системы позволяют использовать подходящие содержательные имена, например: RESPONSES (ответ), TOTALS (итог) и ADDRESSESS (адреса) . Как и в случае числовых массивов, в программе надо указывать оператор DIM, задающий максимальное значение индекса. В системе не должно существовать никакой связи между строковым массивом и одноименным числовым массивом.

Уточним понятия массива строк: каждый его элемент равноценен строковой переменной и может содержать от нуля до большого числа символов. (Предельные значения числа символов приведены в табл. 4.1.) Элементами массивов строк можно пользоваться в сочетании со всеми строковыми функциями, обсуждавшимися в предыдущих разделах.

Одномерный массив строк можно представлять себе в виде списка элементов, например списка фамилий в телефонном справочнике или списка покупок. Приведенная ниже тривиальная программа читает и затем печатает элементы такого списка:

```
10 REM ДЕМОНСТРАЦИЯ РАБОТЫ С МАССИВОМ СТРОК СИМВОЛОВ
20 DIM A$< 10)
30 FOR I=0 TO 3
40   INPUT A$(I)
50 NEXT I
60 REM
70 PRINT
80 PRINT "ВЫВОД ИЗ МАССИВА В ОБРАТНОМ ПОРЯДКЕ" .
90 FOR I=3 TO 0 STEP -1 100   PRINT A$(I) 110 NEXT I 120 END
RUN
?1 ФУНТ КАРТОФЕЛЯ
?2 УНЦИИ ГРИБОВ
?1 КАБАЧОК
?6 ЯБЛОК
ВЫВОД ИЗ МАССИВА В ОБРАТНОМ ПОРЯДКЕ
6 ЯБЛОК
1 КАБАЧОК
2 УНЦИИ ГРИБОВ
1 ФУНТ КАРТОФЕЛЯ END AT LINE 120
```

В силу обстоятельств могут понадобиться двумерные массивы строк и массивы строк большего числа измерений. Не забывайте, что многомерные массивы занимают в памяти много места, особенно массивы строк символов, в кото-

рых каждый элемент может содержать большое число символов, каждому из которых требуется один байт памяти. Массиву A$(9,9,9) при среднем числе символов по 20 на каждый элемент потребуется не менее 20* 10* 10* 10 = = 20 000 байт памяти!

4.3.1. ПРИМЕР ПРОГРАММЫ ДЛЯ РАБОТЫ СО СТРОКАМИ СИМВОЛОВ

Обсуждаемая здесь программа преобразует десятичное число в римское. Форма записи римских чисел обусловливает необходимость использования строк символов. Решение задачи отнюдь не состоит в простой замене десятичных цифр римскими цифрами вследствие разных принципов построения этих двух систем счисления. Например,

1025 преобразуется в MXXV 1982 преобразуется в MCMLXXXII 2000 преобразуется в MM

Десятичная система основана на значении, определяемом позицией цифры в числе, а система римских чисел - на сложении (и вычитании) значений цифр. Существует правило повторения одной цифры до трех раз, так что III равно 3, а XX 20. Существует правило вычитания, согласно которому меньшее значение вычитается из стоящего справа большего значения, например XL равно 40. Существует правило сложения, согласно которому значения цифр складываются с учетом указанных выше правил, например CXLIII равно 100 + 40 + 3 = 143. Десятичные эквиваленты римских цифр таковы:

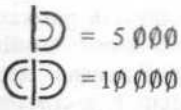
1= 1 V=5
X=10 L= 50 C=100 D=500 M=1000

Необходимость комбинировать указанные три правила усложняет программу ввиду того, что ей надо следить за изменениями в расположении цифр. Запишем последовательность римских чисел от 1 до 10:

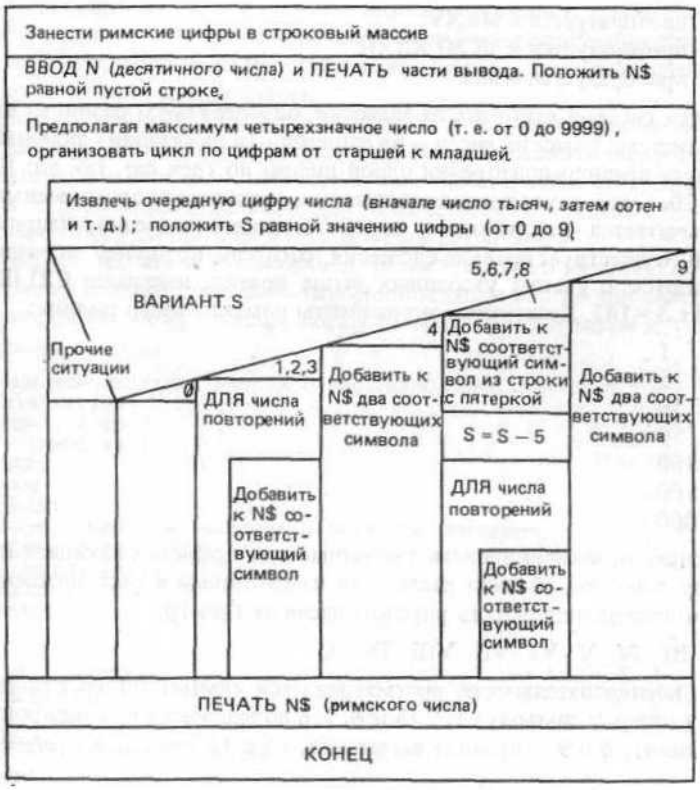
I II III IV V VI VII VIII IX X

На этой последовательности просматривается симметрия по отношению к каждому новому символу. 1, 2, 3 и 6, 7, 8 подчиняются правилу повторения (и сложения), 4 и 9 — правилу вычитания, а 5 и 10 являются новыми символами.
Массив строк символов применяется как таблица для поиска римских цифр, ему присваиваются следующие значения:
I X C M
V L D пустая строка
X C M пустая строка

Первый столбец используется для десятичных значений от 0 до 9, второй — от 10 до 90, третий - от 100 до 900, а четвертый - для 1000 и больших чисел. Так как символы для обозначения больших чисел трудно напечатать, то эта программа не сможет преобразовать числа, большие 3999. Однако при желании в четвертый столбец можно включить следующие символы для больших значений:



Ниже приводится структограмма этой программы:



В основном цикле из введенного десятичного числа извлекаются цифры: вначале - число тысяч, затем - сотен, затем - десятков и в заключение - единиц. Внутри цикла присутствует новый элемент, служащий примером оператора выбора. В Бейсике он называется ON-GOTO. Этот оператор похож на расширенный оператор IF, отсюда и сходное начертание символа структограммы. Вначале вычисляется выражение, указанное после ON; оно округля-

ется до целого числа, значение которого используется для выбора одного из номеров строк, указанных после GOTO. Если вычисление выражения привело к значению 1, то выбирается первый номер строки, к значению 2 — второй номер строки, 3 — третий и т. д. Затем управление передается к строке с выбранным номером.

Оператор ON-GOTO

Общая форма записи:

ON e GOTO $s_1,s_2,s_3,. . . ,s_n$

где e *— арифметическое выражение, а* s_1 — s_n *- номера операторов (по крайней мере, один должен быть указан).*

Действие оператора состоит в передаче управления (GOTO) *одному из нескольких операторов в зависимости от значения выражения* e, *которое вычисляется и затем округляется до ближайшего целого числа.*

Если это целое число равно 1, то управление передается оператору с первым номером (считая слева), т. е. с номером s_1; *если целое число равно 2, то управление передается оператору с номером* s_2 *и т. д.*

Если целое число лежит вне допустимого диапазона (1... n), то управление передается оператору, следующему за оператором ON-GOTO.

Простой иллюстрацией служит программа

```
10  REM ДЕМОНСТРАЦИЯ ПРИМЕНЕНИЯ ОПЕРАТОРА ON-GOTO
20  K=-1
30  PRINT "K=";K; "И ИСПОЛНЯЕТСЯ СТРОКА";
40   ON K GOTO 70,90,110
50   PRINT " 50"
60  GOTO 120
70   PRINT " 70"
80   GOTO 120
90   PRINT " 90"
100   GOTO 120
110   PRINT " 110"
120 K=K+1
130 IF K<=4 THEN 30
140 END
RUN
K=-1   И ИСПОЛНЯЕТСЯ СТРОКА  50
K=0  И ИСПОЛНЯЕТСЯ СТРОКА  50
K=1   И ИСПОЛНЯЕТСЯ СТРОКА   70
K=2   И ИСПОЛНЯЕТСЯ СТРОКА   90
K=3   И ИСПОЛНЯЕТСЯ СТРОКА   110
K - 4  И ИСПОЛНЯЕТСЯ СТРОКА  50 END AT LINE 140
```

Каждая альтернатива должна заканчиваться оператором GOTO "конец группы", чтобы избежать продолжения исполнения в другой альтернативе. Если результат вычисления выражения выходит за пределы числа номеров операторов в списке после GOTO, то исполнение программы продолжается с оператора, следующего за оператором ON-GOTO, в данном случае —со строки 50. Полная программа преобразования такова:

```
10 REM ПРОГРАММА ПРЕОБРАЗОВАНИЯ ДЕСЯТИЧНОГО ЧИСЛА В РИМСКОЕ
20 REM ДОПУСКАЕТСЯ ВВОД ЧИСЕЛ ОТ 0 ДО 3999
```

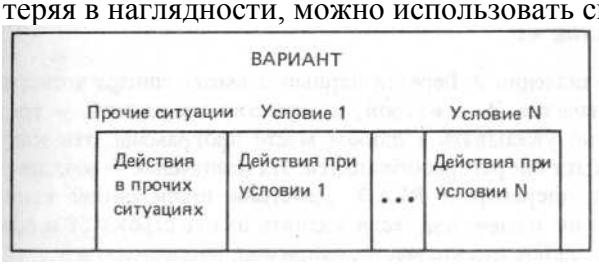
```
30 DIM R$(3,3)
40 REM ЗАПОЛНЕНИЕ МАССИВА ЗНАЧЕНИЯМИ
50 R$ ( 0,1)="I"
60 R$ (0,2)="V"
70 R$(0,3)="X"
80 R$(1,1)="X"
90 R$(1,2)="L"
100 R$(1,3)="C"
110 R$(2,1)="C"
120 R$(2,2)="D"
130 R$(2,3)="M"
140 R$(3,1)="M"
150 REM
160 INPUT N
170 PRINT "ДЕСЯТИЧНОЕ";N;"РИМСКОЕ";
180 N$=""
190 FOR J=3 TO 0 STEP -1
200  T=10**j
210       S=INT(N/T)
220   N=N-S*T
230   ON S+1GOTO 430,260,260,260,310,340,340,340,340,410
240 REM ПО УМОЛЧАНИЮ - ХОЛОСТОЙ ПРОХОД
250   GOTO 430
260 REM S = 1, 2, 3
270   FOR K=1 TO S
280     N$=N$+R$(J,1)
290   NEXT K
300   GOTO 430
310 REM S=4
320   N$=N$+R$(J,1)+R$(J,2)
330   GOTO 430
340 REM S = 5, 6, 7, 8
350   N$=N$+R$(J,2)
360   S=S-5
370   FOR K=1 TO S
380     N$=N$+R$ (J,1)
390   NEXT K
400   GOTO 430
410 REM S=9
420   NS=N$+R$(J,1)+R$(J,3)
430 REM КОНЕЦ ON-GOTO
440 NEXT J
450 PRINT N$
460 END
```

Не поддайтесь искушению использовать симметрию действий для чисел от 1 до 4 и от 5 до 9 посредством беспорядочного применения операторов GOTO. Подобное усложнение программы обесценивает возможную выгоду. Позже в одной из глав будет показано, как можно применить подпрограммы и
функции в такой ситуации.

На основе приведенной выше программы выполните следующее упражнение: после строки 170 преобразуйте N в строковое значение (назовите его M$) , затем замените строки 200-220 на извлечение очередной цифры как символа строки M$, после чего не забудьте преобразовать цифру в число, требуемое для оператора ON-GOTO в строке 23(5. Результаты должны быть те же, что и до внесения изменений.

Общая форма оператора выбора не реализована в Бейсике; ее можно проиллюстрировать следующим символом:

Во всяком случае, Бейсик позволяет сконструировать упрощенный вариант оператора выбора за счет тщательного сочетания оператора ON-GOTO и операторов GOTO, указываемых в конце каждой альтернативы. Недостатками приведенного выше символа является его громоздкость и то, что для написания текста в прямоугольниках процессов выделяется неодинаковое пространство. Несколько теряя в наглядности, можно использовать символ



но с увеличением N он растет вширь, и иногда его приходится преобразовывать к виду

который может ввести в заблуждение тем, что вопреки принятому для нормальной диаграммы правилу действия в прямоугольнике процесса не продолжаются действиями низлежащего

прямоугольника. Две последние формы символа оператора выбора не будут использоваться в данной книге.

4.4. ОПЕРАТОРЫ READ И DATA

Оператор DATA является средством запоминания ограниченного числа данных в самой программе и имеет определенные преимущества по отношению к присваиванию значений констант оператором LET. Данные извлекаются из операторов DATA оператором READ. В качестве примера приведем следующую программу".

```
10  REM ДЕМОНСТРАЦИЯ ДЕЙСТВИЯ ОПЕРАТОРОВ READ-DATA
20  READ A, B, C
30  PRINT  A, B, C
40  REM
50  DATA  3, -2, 45
60  END
RUN
3                -2               45
END  AT  LINE  60
```

В качестве значения A берется первый элемент списка в операторе DATA, в качестве значения B — второй, в качестве значения C — третий. Оператор DATA можно указывать в любом месте программы, так как при передаче управления эти операторы обходятся. Их назначение — создать запас значений для чтения оператором READ. Действия приведенной выше программы совершенно не изменятся, если удалить из нее строку 50 и поместить оператор DATA в любое другое место, например, под номером 5.

Так как в программе допускается несколько операторов DATA, то для удобства чтения рекомендуем группировать все операторы DATA в начале или в конце программы.

4.4.1. УПОТРЕБЛЕНИЕ ОПЕРАТОРОВ DATA

Вообразите, что существует указатель, показывающий на самое начало списка данных в операторе DATA. В приведенном выше примере оператор READ присваивает переменной A первое число из списка данных в операторе DATA (т. е. 3) и перемещает указатель на следующее число (—2) .

Каждое действие в операторе READ состоит в присваивании переменной того значения, на которое показывает текущий указатель. Так, если указатель показывает на —2 и очередной переменной является B, то этой переменной будет присвоено значение —2 и указатель переместится на одну позиции).

Если указатель достиг конца оператора DATA и затребованы оператором READ новые данные, то либо указатель перемещается на начало списка дан-

ных следующего оператора DATA, либо, если такого оператора нет, выдается сообщение о нехватке данных, например:

```
10  READ A,B,C,D,E,F,G,H,I
20  DATA 14,32,96,78,3,11
30  DATA 303,411,9999
```

Хотя операторы DATA можно указывать в любом месте программы, полезно группировать их вместе, так как они связаны в том смысле, что как только оператор READ покончит с одним оператором DATA, он примется за следующий оператор DATA с большим номером. Очень распространенная ошибка состоит в несовпадении между числом значений, требуемых одним или несколькими операторами READ, и числом значений, распределенных по многим операторам DATA. Такую ошибку гораздо легче выявить, если все операторы DATA собраны вместе.

Операторы READ и DATA полезны для хранения в программе данных, не изменяющихся от запуска к запуску. Они также очень полезны для присваивания начальных значений переменным или массивам. Обратимся к программе преобразования числа из подразд. 4.3.1: используя операторы READ и DATA, строки 50—140 можно заменить на цикл

```
50   FOR K=0 TO 3 60      FOR L = 1 TO 3 70        READ R$(K,L)
80     NEXT L
90    NEXT К
```

дополненный операторами

```
100  DATA I,V,X,X,L,C,C
110   DATA D,M,M,,,
```

Учтите, что, как и в случае оператора INPUT, при указании строковых значений кавычки не обязательны и оператор

50 DATA "ФРЕД","ДЖИМ" имеет тот же эффект, что и оператор 50 DATA ФРЕД, ДЖИМ

Обычно строка заключается в кавычки только в случае, если в нее должна входить запятая. Так, оператор

50 DATA "ФРЕД, ДЖИМ" задает одно строковое значение ФРЕД, ДЖИМ.

Операторы READ и DATA

Общая форма записи: READ *переменная1, переменная2,.. .*

Оператор READ *последовательно присваивает очередные значения из списка данных оператора* DATA *переменным: переменшя1, переменная2 и т. д.*

Общая форма записи: ,

DATA *значение!, зшчение2,...*

В операторе DATA *через запятую указываются константы (как числовые, так и строковые), значения которых присваиваются переменным, перечисленным в операторе* READ.

4.4.2. ОПЕРАТОР RESTORE

Воображаемый указатель на значение из списка данных оператора DATA можно вновь установить на начало списка первого оператора DATA с помощью оператора RESTORE (восстановление), который можно указать в любом месте программы. Для изучения его действия изменим пример программы, данный в начале разд. 4.4, добавляя строки 12, 25, 28 и изменяя строку 30 так, чтобы получилась следующая программа:

```
I0 REM ДЕМОНСТРАЦИЯ ДЕЙСТВИЯ ОПЕРАТОРОВ READ-DATA
12 REM И RESTORE
20 READ A,B,C
25   RESTORE
28   READ D,E,F
30 PRINT A,B.C,D,E,F
40 REM
50 DATA 3,-2,45
60 END
RUN
3                -2               45               3
-2               45
END AT LINE 60
```

Указанный в ней оператор RESTORE позволяет новому оператору READ в строке 28 заново прочитать значения, перечисленные в операторе DATA на
строке 50.

Приведенная ниже программа иллюстрирует практическое приложение операторов READ, DATA и RESTORE; она запоминает фамилии, адреса и распечатывает адрес, соответствующий введенной фамилии. Обратите внимание на использованный в структограмме символ "ПОВТОРЯТЬ ДО": аналогичная структура уже применялась в программе из подразд. 4.2.5. В программе должны быть учтены две возможности: либо достигается конец списка фамилий и адресов в операторе DATA, либо в этом списке обнаруживается фамилия, совпадающая с введенной. В последнем случае переменная F полагается равной 1 и исполнение цикла завершается.

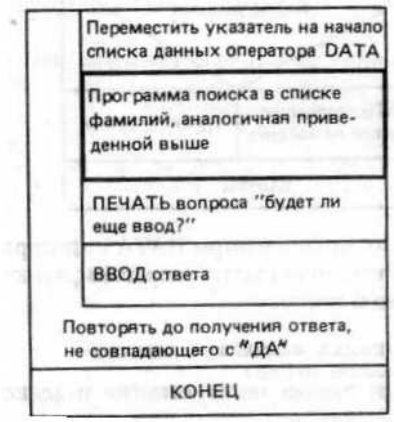
Обратите внимание на то, что операторы DATA на диаграмме не изображены: в структограммах изображаются только операторы, выполняющие какие-либо действия. Сама программа такова:

```
10 REM  ПРОГРАММА ПОИСКА ФАМИЛИЙ И АДРЕСОВ
20 REM   (В ОГРАНИЧЕННОМ ЧИСЛЕ)
30 REM  Р ПОЛАГАЕТСЯ РАВНЫМ ЧИСЛУ ФАМИЛИЙ И АДРЕСОВ 40 REM
50 P=5
60 PRINT "ВВЕДИТЕ ФАМИЛИЮ";
70 INPUT N$
80 I=0
90 F=0
100   I=I+1
110   READ A$.B$
120   IF N$=A$ THEN F=1
130   IF I<P AND F=0 THEN 100
140 IF F=1 THEN 170
150 PRINT "АДРЕС НЕ ИЗВЕСТЕН"
160 STOP
```

```
170 PRINT B$
180 DATA УИЛСОН,126 PALACE ROAD LONDON
125


190 DATA АРДЕН.5 OXFORD STREET LIVERPOOL
200 DATA ЭЛТИ.94 WATER STREET WALLASEY
210 DATA ДЖОНС,2 SCOTLAND ROAD BRIGHTON
220 DATA СМИТ,42 KEITH PLACE YORK
230 END
RON
ВВЕДИТЕ ФАМИЛИЮ ?СМИТ
42 KEITH PLACE YORK
END AT LINE 230
RUN
ВВЕДИТЕ ФАМИЛИЮ ? УОЛТОН
АДРЕС НЕ ИЗВЕСТЕН
END AT LINE 230
```

Для удобства чтения каждая пара фамилия-адрес указана в отдельном операторе DATA, но такая запись не обязательна. Поскольку фамилии и адреса никак не упорядочены, то в программе применен самый элементарный метод поиска. При небольшом числе данных (менее 100 элементов) метод поиска не имеет особого значения, так как время полного просмотра списка данных невелико.

Запускать программу заново для получения новой справки — скучное занятие. Гораздо лучше организовать цикл, изображаемый следующей структограммой:



При реализации внутреннего прямоугольника приведенную выше программу надо слегка изменить для удаления операторов STOP и END. Текст этих и остальных изменений таков:

```
5 RESTORE
160 GOTO 230
230 НЕМ КОНЕЦ ПОИСКА
240 PRINT "ПРОДОЛЖИТЬ";
250 INPUT A1$
260 IF A1$="ДА" THEN 5
270 END
```

Действие оператора RESTORE состоит в перемещении перед началом каждого поиска воображаемого указателя значения на начало списка данных первого оператора DATA. Между применением операторов READ-DATA и работой с файлами данных существует тесная взаимосвязь. Чтобы преобразовать программу для работы с последовательным файлом данных, в нее требуется внести очень небольшие изменения, если не считать добавления нескольких инструкций или команд, обеспечивающих доступ программы к файлам. Например, в системе с Бейсиком для ЭВМ ICL 2903/4 некоторые операторы для работы с файлами при указании специальных параметров имеют тот же эффект, что и операторы READ, DATA и RESTORE.

Таким образом, использование операторов READ-DATA служит введением в работу с файлами. В некоторых языках программирования, например в Паскале, файлы рассматриваются как еще один тип "объектов", которыми может манипулировать программа, и для выполнения этих манипуляций существуют особые операторы. Полное описание файлов дается в гл. 8, а только что сделанные комментарии завершают обзор "объектов", которыми программист может воспользоваться при работе с Бейсиком.

4.5. СОРТИРОВКА И ПОИСК

На темы сортировки и поиска написано очень много литературы. Настоящий раздел может служить не более чем введением с целью очертить некоторые основные направления и дать достаточно хорошие примеры методов общего назначения. Мы будем предполагать, что данные содержатся во внутренней памяти в массиве, что подразумевает наличие небольшого или умеренного числа данных. Большие файлы данных, хранящиеся на магнитных лентах или дисках, требуют других методов сортировки и поиска; представление об этих методах дается в гл. 8.

Сортировкой является такая перестановка элементов (в массиве), после которой они оказываются упорядоченными требуемым образом. Примером сортировки служит упорядочение по алфавиту.

Поиск заключается в отыскании элемента (в массиве), значение которого совпадает с ключом поиска. Примером поиска служит отыскание элемента данных, включающего фамилию и адрес, по заданной фамилии.

Вначале рассмотрим сортировку, так как эффективный поиск опирается на то или иное упорядочение данных. Если данные никак не упорядочены, то единственным способом поиска требуемого значения является последовательная проверка каждого элемента данных, что и было продемонстрировано на примере в подразд. 4.4.1.

Общую задачу сортировки данных в массиве можно свести к выбору методов, каждый из которых требует различного времени исполнения и использует различное число дополнительных рабочих ячеек памяти. При работе с массивом заданного размера приходится идти на компромисс между временем исполнения и числом рабочих ячеек: чем быстрее надо выполнить сортировку, тем больше требуется рабочих ячеек.

4.5.1. СОРТИРОВКА МЕТОДОМ ПУЗЫРЬКА

Этот метод очень прост; он практически не требует рабочих ячеек, но работает довольно медленно. Однако его легко понять и он может служить идеальным методом сортировки общего назначения в условиях, когда память ЭВМ ограничена, а время исполнения не очень существенно.

Этот метод основан на попарном сравнении смежных элементов данных; если порядок следования элементов в очередной паре неправилен, то эти элементы обмениваются местами. Для выполнения обмена требуется дополнительная переменная, сохраняющая на время обмена одно из значений. Если значения, которыми надо обменяться, содержатся в переменных A и B, то обмен можно выполнить посредством операторов

```
10 T=A 20 A = B 30   B=T
```

использующих переменную T (в качестве рабочей ячейки). Обмен значениями строковых переменных осуществляется аналогичным образом.
Обсудим сортировку числовых значений в порядке убывания. Применение метода пузырька к массиву с размером 4 иллюстрирует рис. 4.2:
(а) Пройдитесь по массиву от первого элемента до последнего, выбирая по очереди пару смежных значений. Если левое значение меньше правого, то выполните обмен значениями. Это гарантирует перемещение наименьшего значения в последний элемент массива, хотя остальные значения не обязаны

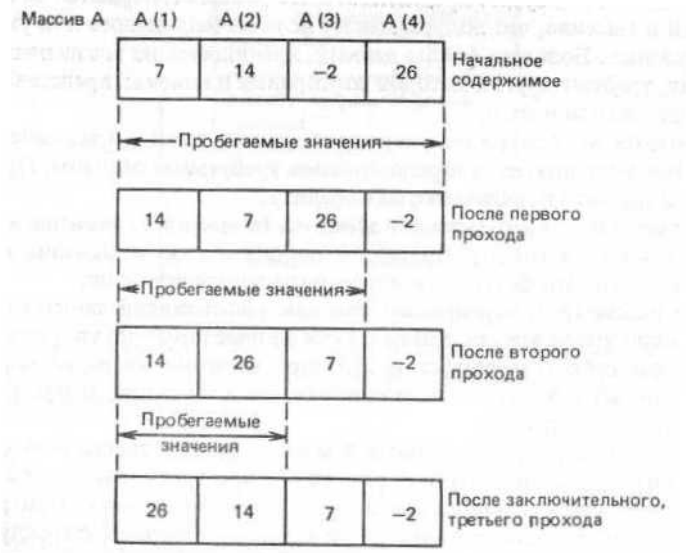
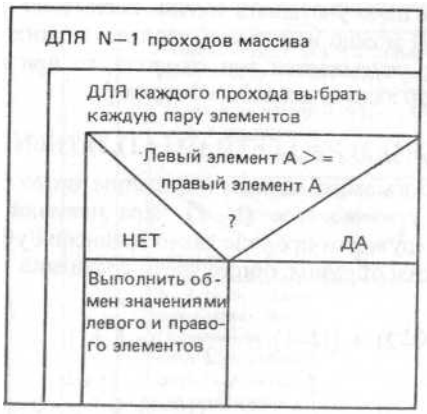


Рис. 4.2. Стадии пузырьковой сортировки массива из четырех значений

стать упорядоченными. Теперь ограничьтесь оставшимися значениями, находящимися в элементах от первого до предпоследнего.
(б) Примените процедуру (а) к сократившемуся множеству значений, находящихся в элементах от первого до предпоследнего. Как и при первом проходе, это гарантирует перемещение наименьшего значения в рассматриваемом множестве в свою конечную позицию — в предпоследний элемент массива.
(в) Рассмотрите оставшиеся значения и продолжайте действовать аналогичным образом.
Таким образом, при применении этого метода каждый следующий проход становится все короче и значения занимают свои места по направлению от конца массива к его началу. Каждый проход обеспечивает перемещение наименьшего значения в конец рассматриваемой порции массива, т. е. "подъем" вверх в правильную позицию.
Ниже приводится структограмма, изображающая метод пузырька, а также фрагмент программы:




```
1000 REM ФРАГМЕНТ ПРОГРАММЫ ПУЗЫРЬКОВОЙ СОРТИРОВКИ
1010 REM ПРЕДПОЛАГАЕТСЯ, ЧТО ЗНАЧЕНИЯ СОДЕРЖАТСЯ
1020 REM В МАССИВЕ A(1), .... A<N) И СОРТИРУЮТСЯ
1030 REM В ПОРЯДКЕ УБЫВАНИЯ
1040 FOR I=N TO 2 STEP -1
1050   FOR J=1 TO I-1
1060     IF A(J)>=A(J+1) THEN 1100
1070     T=A(J)
1080     A(J)=A(J+1)
1090     A(J+1)=T
1100   NEXT J
1110 NEXT I
1120 REM КОНЕЦ СОРТИРОВКИ
```

Для сортировки данных в порядке возрастания надо попросту изменить Условие в строке 1060 на <=. Строковые данные можно сортировать в строковом массиве A$, используя рабочую строковую переменную T$.

Иногда с каждым логическим элементом данных ассоциируется несколько переменных, скажем фамилия в A$ и адрес в B$; пусть, к примеру, эти массивы описаны оператором

```
10   DIM A$(95),B$(95)
```

Сортировка по фамилиям в AS осуществляется так, как описано выше, однако теперь в программу требуется добавить операторы, обеспечивающие выполнение в B$ в точности тех же перемещений, которые происходят в AS. Для этого строки 1060 - 1090 надо заменить на операторы

```
1060          IF  A$(J)>=A$( J+1)   THEN   1100
1065         T$=A$(J)
1070        A$( J)=A$(J+l)
1075        A$(J+1)=T$
1085          T$=B$(J)
1085          B$(J)=B$(J+l)
1090          B$(J+1)=T$
```

Если при сортировке надо учитывать только несколько первых букв фамилии, то в строке 1060 можно выполнять извлечение этих символов и сравнение. Например, если учитывается три символа, то при работе с Бейсиком Microsoft строку 1060 надо заменить на оператор

```
1060  IF LEFT$(A$(J),3) >= LEFT$(A$(J + 1),3) THEN 1100
```

Чтобы оценить скорость этого метода, определим число сравнений значений. На первом проходе сравнивается (N-1) пара значений, на втором (N-2) пары; на каждом следующем проходе число сравнений убывает на 1, пока не станет равным 1. Таким образом, общее число сравнений

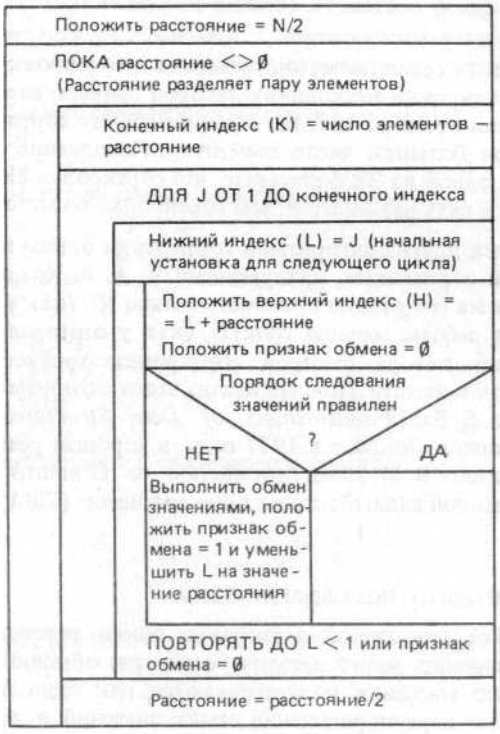
$$1+2+3+\ldots+(N-2)+(N-1)=\frac{N-1}{2}(N-1+1)$$
$$=\frac{N}{2}(N-1)\simeq\frac{N^2}{2}\ .$$

Если в определенной части случаев, скажем в половине, за этими сравнениями последует обмен значениями, то число обменов окажется приблизительно равным $N^2/4$. Поэтому полное время выполнения сортировки пропорционально N^2, где N - число сортируемых элементов данных.

4.5.2. ДРУГИЕ МЕТОДЫ СОРТИРОВКИ

Для столь же длительного описания других методов сортировки просто не хватит места. Поэтому мы ограничимся краткими комментариями по поводу их действия. Полные описания методов сортировки можно найти во многих книгах, посвященных алгоритмам для ЭВМ. Особо рекомендуем книгу Peter Naur *Concise Survey of Computer Methods,* выпущенную издательством Studentliteratur в Лунде (Швеция) в 1974 году.

Сортировка Шелла быстрее метода пузырька и требует выполнения $N\log_2(N)$ операций, где N — число сортируемых элементов данных. Она похожа на метод пузырька, но, в отличие от него, начинает сравнивать не смежные, а далеко отстоящие друг от друга значения (примерно на N/2) и сортирует все эти значения, а затем уменьшает расстояние между сравниваемыми значениями. На последнем проходе расстояние между ними равно 1 и поэтому фактически этот проход выполняется по методу пузырька. Такая сортировка предложена Д. А. Шеллом и основана на процедуре Дж. Бутройда, написанной на языке Алгол 60.



Этой структограмме отвечает следующий фрагмент программы:

```
1000   REM СОРТИРОВКА ШЕЛЛА
```

```
l010 REM ПРЕДПОЛАГАЕТСЯ, ЧТО В МАССИВЕ A$(1).....A$(N)
1020   REM СОДЕРЖАТСЯ СТРОКИ СИМВОЛОВ. КОТОРЫЕ
1030   REM НАДО ОТСОРТИРОВАТЬ ПО АЛФАВИТУ
1040   D=N/2
1050       REM
1060        IF D=0 THEN 1210
1070            K=N-D
1080            FOR J=1 TO K
131



1090               L=J
1100                   H=L+D
1110                   S=0
1120                   IF A$(L)>=A$(H) THEN 1180
1130                   T$=A$(L)
1140                   A$(L)=A$(H)
1150                   A$(H)=T$
1160                   L=L-D
1170                   S=1
1180                IF L=1 AND S=1 THEN 1100
1190            NEXT J
1195            D=INT(D/2)
1200       GOTO 1060
1210        REM ВЫХОД С ОТСОРТИРОВАННЫМ МАССИВОМ
```

Если Вы хотите сравнить метод пузырька и сортировку Шелла, то примените их для сортировки нескольких наборов данных, варьируя число сортируемых значений от 10 до 100. Вы обнаружите, что сортировка Шелла гораздо быстрее при большем числе элементов. Исполнение приведенной выше программы на одной из ЭВМ показало, что сортировка Шелла отсортировала 100 элементов в пять раз быстрее, чем сортировка по методу пузырька.

Среди многих других алгоритмов сортировки одним из лучших является метод быстрой сортировки, придуманный Ч. А. Р. Хоаром. Хотя в наихудшем случае время его работы пропорционально N^2 (как у метода пузырька) , среднее время работы меньше $N\log_2 N$ (как у сортировки Шелла). Однако для реализации метода быстрой сортировки требуется дополнительная рабочая область в памяти. Полный анализ этого алгоритма содержится в книге E. Horowitz, S. Sahi *Fundamentals of Data Structures,* выпущенной издательством Pitman в Лондоне в 1977 году, а хорошая реализация на Бейсике приводится в книге R. M. Jones *Introduction to Computer Applications Using BASIC,* выпущенной издательством Allyn and Bacon (США) в 1981 году.

4.5.3. СОРТИРОВКА С ПОМОЩЬЮ ИНДЕКСА

Требующийся для любой сортировки обмен значениями на некоторых ЭВМ может занимать много времени, если надо обмениваться строками или если несколько массивов рассматриваются как один логический элемент. Однако вместо переупорядочения самих значений в процессе сортировки можно образовать индекс (предметный указатель), в котором отмечаются правильные места значений в массиве. Во время сортировки значения остаются на исходных местах, а изменяется индекс. По окончанию сортировки индекс используется для копирования сортируемых значений в новый массив или служит справочником для работы с исходным массивом.

Если нам требуется отсортировать массив A(100) , то надо ввести индекс I(100) , лучше всего целочисленный, если такая возможность предусмотрена в системе (это сократит занимаемую массивом память), и задать начальные значения его элементов операторами

FOR L = 0 TO 100
I(L) =L NEXT L
Для индекса должно выполняться соотношение I (исходная позиция) = новая позиция
поэтому перед началом сортировки I(1)=1, I(2)=2 и т. д. При сортировке во время каждого прохода сравниваются значения из A, определенные по I( ) , а переставляются значения индекса в I( ) .
При введении индекса программа для сортировки методом пузырька из подразд. 4.5.1 превратится в следующую программу:

```
  900 REM СОРТИРОВКА С ПРИМЕНЕНИЕМ ИНДЕКСА
  910 FOR L=1 TO N
  920   I(L)=L
  930 NEXT L
 1000 REM ФРАГМЕНТ ПУЗЫРЬКОВОЙ СОРТИРОВКИ
 1010 REM
 1020 REM
 1030 REM
 1040 FOR I=N TO 2 STEP -1
 1050   FOR J=1 TO I-1
 1052     I1=I(J)
 1054     I2=I(J+1)
 1060     IF A(I1)>=A(I2) THEN 1100
 1070       I(J)=I2
 1080       I(J+1)=I1
 1100   NEXT J
 1110 NEXT I
 1120 REM КОНЕЦ СОРТИРОВКИ С ПРИМЕНЕНИЕМ ИНДЕКСА
```

Операторы

```
FOR L = 1 TO N
  PRINT A (L);
NEXT L
```

распечатают исходный, неизмененный набор значений, а операторы FOR L = 1 TO N
PRINT A (I1) NEXT L
распечатают отсортированный набор значений.
Применение индекса дает большое преимущество в случае, когда значения данных распределены по нескольким массивам. Если, скажем, сортируются значения массивов A (10), B(9), C(27), то можно создать индекс I(46), в котором первые 10 элементов указывают на массив A, следующие 9 — на массив B, а последние 27 — на C. Конечно, алгоритм надо модифицировать так, чтобы в сравнениях могли участвовать как элементы массива A, так и элементы массивов B и C.

При аккуратном подходе аналогичным образом можно модифицировать и другие методы сортировки. Кроме того, можно предложить особые приемы ускорения процесса сортировки за счет использования индекса.

4.5.4. ПОИСК

Поиском заданного значения в массиве приходится заниматься очень часто. При этом задается аргумент поиска и требуется определить положение в массиве такого элемента, у которого значение ключа совпадает с аргументом. Если порядок расположения данных в массиве неизвестен, то нет более эффективного метода, чем описанный ранее простой последовательный поиск, при котором ключ каждого элемента данных сравнивается с аргументом поиска некоторым регулярным образом. При этом для поиска в массиве из N значений в среднем приходится выполнить N/2 сравнений.
Предложить методы более быстрого поиска можно только в том случае, если данные определенным образом упорядочены. Например, тот факт, что в телефонном справочнике записи упорядочены по алфавиту, позволяет нам очень быстро находить нужную фамилию и номер телефона.
Универсальным методом быстрого поиска в упорядоченном множестве данных является бинарный поиск, при котором число сравнений, выполняемых для поиска в массиве из N значений, в среднем

пропорционально $\log_2 N$. Сравним это число со средним числом сравнений в простом методе последовательного поиска, которое пропорционально N:

N	10	100	1000
$\log_2 N$	3.3	6.6	9.9

Очевидно, дополнительные затраты на упорядочение множества данных более чем окупаются при применении бинарного поиска.

Метод бинарного поиска заключается в последовательном делении множества данных на две равные части — отсюда и название "бинарный" (двоичный) . Первоначально областью поиска считается все множество; из него извлекается средний элемент, ключ которого сравнивается с аргументом поиска. Если они совпали, то поиск на этом успешно завершается, но если они не совпадают, то аргумент поиска может быть больше или меньше ключа среднего элемента. Тогда в зависимости от способа упорядочения данных в качестве области поиска выбирается первая или вторая половина множества, из нее извлекается средний элемент и его ключ сравнивается с аргументом поиска. Этот процесс продолжается до тех пор, пока не приведет либо к /спеху, либо к неудаче. Такое последовательное деление множества работает на удивление хорошо. На рис. 4.3 показан первый шаг поиска в массиве

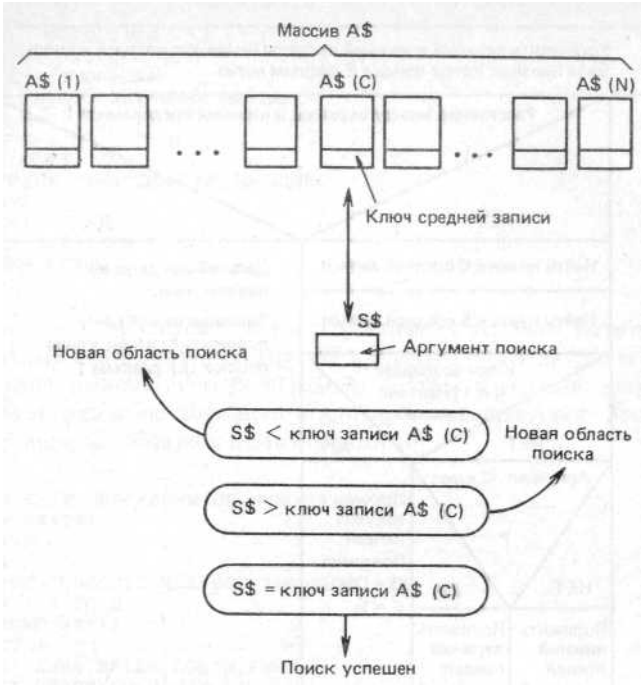
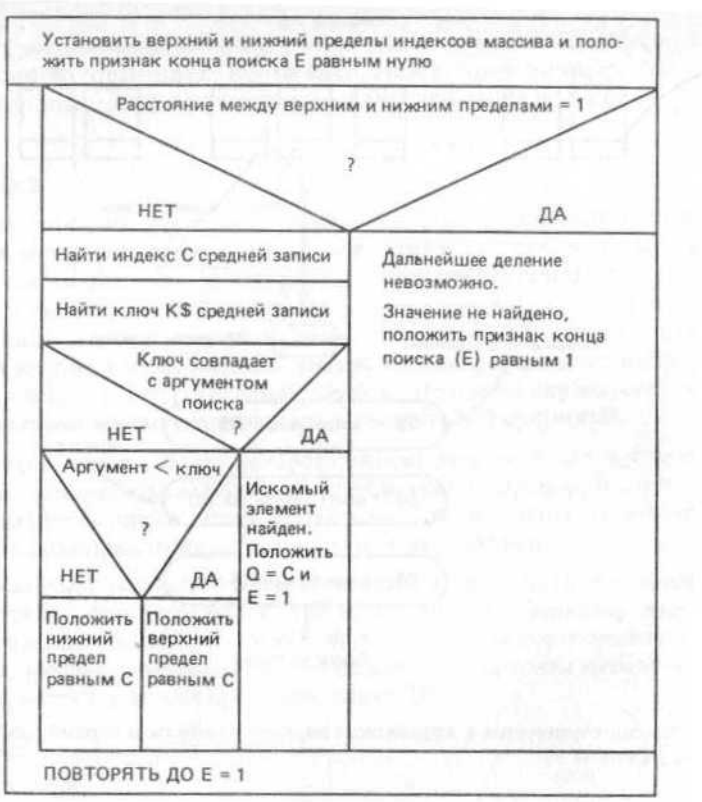


Рис. 4.3. Массив сортируется в алфавитном порядке. Изображен первый цикл двоичного поиска с ключом S$

строк символов, упорядоченных по алфавиту по нескольким первым символам каждой строки (являющихся ключом).

Приведенная ниже форма алгоритма бинарного поиска следует форме, предложенной П. Науром, согласно которой вводятся фиктивные нижняя и верхняя границы значений индексов массива, равные -1 и N+1, в то время как фактически индексы пробегают значения от 0 до N. В этом алгоритме первые M символов каждой строки рассматриваются как ключ и строки упорядочиваются в алфавитном порядке по этим ключам. Строки могут содержать фамилии, за которыми следуют адреса или какая-либо другая информация; таким образом, процедура поиска может быть частью справочной системы. Структограмма алгоритма такова:



Входными данными для приведенного ниже фрагмента программы служат массив A$(0), .. . , A$(N), значение N, значение M, определяющее число символов в ключе, а также аргумент поиска в переменной S$. На выходе из фрагмента в Q находится индекс найденного элемента массива или N+1, если элемента с заданным значением ключа нет.

```
1000 REM БИНАРНЫЙ ПОИСК
1010 REM ПРЕДПОЛАГАЕТСЯ, ЧТО МАССИВ A$(0), ..., A$(N)
1020 REM СОДЕРЖИТ ЗНАЧЕНИЯ. ОТСОРТИРОВАННЫЕ ПО ПЕРВЫМ M
1030 REM СИМВОЛАМ В АЛФАВИТНОМ ПОРЯДКЕ
1040 REM В Q ВОЗВРАЩАЕТСЯ ЛИБО ИНДЕКС НАЙДЕННОГО ЗНАЧЕНИЯ
1050 REM ЛИБО N+1
1060 B2=-1
1070 T2=N+1
1080 E=0
1090 IF T2-B2=l THEN 1220
1100  C=INT(B2+T2)/2)
1110   K$=LEFT$(A$(C),M)
1120   IF S$=K$ THEN 1160
1130     IF S$<K$ THEN 1160
1140       B2=C
136



1150            GOTO 1250
1160               T2=C
1170            GOTO 1250
```

```
1180      PRINT "ЗНАЧЕНИЕ НАЙДЕНО"
1190      Q=C
1200      E=1
1210      GOTO 1250
1220   PRINT "ЗНАЧЕНИЕ НЕ НАЙДЕНО"
1230   Q=T2
1240   E=1
1250   IF E=0 THEN 1090
1260   REM ВЫХОД
```

Переменная E вводится для учета различных условий выхода из цикла. Произвольные передачи управления за пределы такого цикла не могут быть представлены символами структограммы, поэтому применение структограмм способствует развитию хорошего стиля программирования. Этот фрагмент можно проверить, добавив к нему операторы

```
10 REM ТЕСТ ПРОГРАММЫ ДВОИЧНОГО ПОИСКА
20 DIM A$(2B)
30 N=15
40 M=1
50 INPUT "ВВЕДИТЕ ИСКОМОЕ ЗНАЧЕНИЕ";S$
60 FOR I=0 TO N
70   READ A$(I)
80 NEXT I
90 DATA ABBA, BRIAN, COLIN, FRED
100 DATA GERRY, HUGH, JIM, LEN
110 DATA MIKE, NICK, OLIVER, PETER
120 DATA RUPERT, STAN, TIM, WILLIAM
1300      PRINT Q, A$(Q)
1310   END
```

Запуски этой программы для поиска по очереди всех имен из содержащегося в ней списка к регистрация чисел сравнений показывают, что в среднем на каждый успешный поиск приходится 3,4 сравнения ($\log_2$ 16 равен 4); сами числа распределены между 5 (наихудший случай) и 1 (наилучший случай).

УПРАЖНЕНИЯ

4.1. Напишите программу для чтения десяти чисел из списка данных оператора DATA и печати наибольшего и наименьшего из этих чисел.

4.2. Напишите программу для регистрации и изображения экзаменационных оценок одного класса. Фамилии учеников хранятся в нескольких операторах DATA в программе. Во время исполнения программа по очереди выводит каждую фамилию и запрашивает ввод оценки с терминала. В заключение изображаются средняя оценка и список фамилий с указанием оценок.

4.3. Напишите программу для ведения журнала посещаемости класса. Данные представляются в виде списка единиц и нулей: единица означает, что ученик присутствовал в классе, а нуль — отсутствовал. Для класса из восьми учеников запись одного дня в журнале посещаемости может иметь вид

1, 1,0, 1, 1, 1,0,0 Эти данные можно поместить в оператор DATA или ввести с терминала.

Разработайте программу для чтения сделанных в течение недели (5 дней) записей в журнале посещаемости и печати числа учеников в классе по дням недели, а также числа дней, которые каждый ученик присутствовал в классе (при этом потребуются массивы).

4.4. Разработайте программу для выполнения следующих действий со строками символов :

(а) Присвоить двум строковым переменным соответственно Ваши имя и фамилию, объединить эти строки в одну и распечатать результат.
(б) Выделить из строковых переменных инициалы и напечатать их.
(в) Присвоить одной строковой переменной форму обращения к Вам (Г-Н, Г-ЖА и пр.), а другой - Ваш адрес и распечатать Ваши фамилию и адрес в обычной форме.
4.5. Группа из шести рабочих занята на строительной площадке. В течение каждой недели двое из них отвечают за организацию чаепития. Разработайте и напишите программу для печати такого списка пар имен, составляющего полный график дежурств, в котором каждый из работников отвечает за чаепитие в течение пяти недель. Попробуйте использовать оператор DATA и массив строк символов для работы с фамилиями.
Ваша программа может выдавать справедливый график, при котором все дежурят одно и то же время, но при этом некоторым работникам придется готовить чай несколько недель подряд. Поэтому введите дополнительное ограничение, согласно которому ни один из работников не должен дежурить более двух недель подряд, и разработайте вторую программу, учитывающую это ограничение.
4.6. Разработайте и напишите программу, анализирующую длину слов в предложении. Программа должна воспринимать предложения разного вида, в которых допускаются все обычные знаки препинания. Она должна определять максимальное, минимальное и среднее число букв в словах предложения и печатать полученные результаты.
4.7. Разработайте и напишите программу анализа текста. Она должна воспринимать фрагмент текста как предложение и вычислять частоту появления входящих в него слов. Таким образом, как только при обработке текста встречается новое слово, оно должно запоминаться в таблице со значением частоты появления, равной 1; если обнаруженное слово не является новым, то определяется его положение в таблице и соответствующее ему значение частоты появления увеличивается на 1.
Отсортируйте результаты и изобразите слова в порядке алфавита с указанием частоты их появления.
4.8. Напишите программу для выдачи справок. Соответствующие сведения можно хранить в операторах DATA, предваряя каждую справку "ключевым словом". При исполнении программа должна запросить ввод ключевого слова, найти его положение и распечатать соответствующую ему справку.
В качестве примера используйте часть предметного указателя, рассматривая название предмета в качестве ключа, а номер страницы - в качестве справки. В ответ на ввод названия предмета программа должна выдавать номер страницы, на которой оно встречается.
Можно использовать и любую другую информацию, например по поводу выборов в парламент, рассматривая фамилию депутата в качестве ключа, а число поданных за него голосов - в качестве справки. (Справочную информацию лучше всего помещать в файле, см. гл. 8.)

5. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

Предыдущие главы содержали введение в программирование и описание большинства свойств языка программирования Бейсик, знание которых достаточно для разработки и написания простых программ. В гл. 3 были представлены структограммы, являющиеся превосходным способом наглядного представления эффекта комбинирования небольших групп операторов Бейсика. В сочетании с методом пошаговой детализации, представленным в разд. 3.6, структограммы могут служить также полезным средством разработки программы.

Однако проблема разработки законченной программы для выполнения достаточно сложных действий пока еще не обсуждалась. Попытка разработать программу, не отрываясь от работы за ВТУ, становится более или менее бесплодной, как только программа перестает помещаться на экране дисплея (от 20 до 40 строк) , так как мы склонны забывать то, что уже было написано, и терять управление по мере усложнения ситуации.

По методу пошаговой детализации предлагается делить задачу на небольшие и относительно независимые части, решение которых можно разрабатывать отдельно. Можно ли провести параллель между такими частями, или модулями, и элементами языков программирования? Да, их можно ассоциировать с функциями и подпрограммами, которые может определить сам программист. К сожалению, Бейсик недостаточно полно реализует все те требования, которые необходимо выполнить для полного обеспечения модульности. Тем не менее Бейсик предоставляет вполне приемлемые средства модульного программирования. В некоторых расширениях Бейсика предпринимаются попытки тем или иным путем устранить упомянутые недостатки; в последующих разделах будут описаны как эти расширения, так и стандартные возможности создания функций и подпрограмм.

5.1. ФУНКЦИИ, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ

Для хорошего ведения разработки в первую очередь необходимо иметь возможность объединять группу операторов программы в блок. Такой блок операторов может быть независимым от основной программы и выполнять какую-либо специфичную работу. Например, встроенная в систему функция SQR вычисляет квадратный корень из числа, а вызывается, например, следующим образом:

```
20   Y=5
30   Z = SQR(Y*Y+ 2*Y) или
10   PRINT SQR (981+Z)
```

В предыдущих главах обсуждались стандартные функции (строковые и числовые), которые автоматически производили такие часто требующиеся операции, как извлечение квадратного корня. Если бы эти функции отсутствовали, то пришлось бы самостоятельно разрабатывать и включать в соответствующее место программы набор операторов для вычисления квадратного корня.

При анализе составных частей задачи нередко удается обнаружить некоторые неоднократно встречающиеся действия. Функции и подпрограммы позволяют однажды написать операторы, выполняющие эти действия, и пользоваться ими в нескольких местах программы.

Как функции, так и подпрограммы являются модулями, выполняющими одну или несколько отдельных операций, и предоставляются программе как единое целое. Функции возвращают единственное значение, числовое или строковое, которое присваивается имени функции. Подпрограммы не возвращают значения, но зато могут изменить *любую* переменную или даже *все* переменные, содержащиеся в программе. Подпрограммы не имеют имени.

5.1.1. ОПЕРАТОР DEF FN

Если повторяющиеся действия можно выразить в виде одного оператора Бейсика, результатом исполнения которого является единственное значение (наподобие квадратного корня), то для их выполнения можно определить

функцию.
В стандарте Бейсика допускаются только однострочные функции, обычно помещаемые в начале программы, так как они должны быть определены до того, как будут использоваться. Исключением является Бейсик BBC, в котором функции и процедуры не должны определяться в теле программы. Их лучше всего помещать после завершающего программу оператора END.
Рассмотрим задачу вычисления чистого дохода индивидуума при условии, что сумма, не превышающая 1200 фунтов стерлингов, налогом не облагается, а за каждый фунт сверх этой суммы взимается налог в 35 пенсов:

```
10 REM ЗАДАЧА ПОДСЧЕТА ПРОСТОГО НАЛОГА
20 REM  G=ОБЩИЙ ДОХОД
30 REM  N=ЧИСТЫЙ ДОХОД ПОСЛЕ УПЛАТЫ НАЛОГА
40 DEF FNT(A)=A-(A-1200)*35/100
50 REM
60 INPUT "ВВЕДИТЕ ОБЩИЙ ДОХОД";G
70 IF G<=1200 THEN 100
80   PRINT "ОБЩАЯ СУММА =";G;"4МСТАН ПРИБЫЛЬ =";FNT(G)
90 STOP 100   PRINT "ВЫЧЕТОВ НЕТ. ПРИБЫЛЬ=";G
110 END
RUN
ВВЕДИТЕ ОБЩИЙ ДОХОД ? 4500
ОБЩАЯ СУМКА=4580 ЧИСТАЯ ПРИБЫЛЬ=3345
END AT LINE 90
```

Всякий раз, когда в строке 80 встречается ссылка на функцию FNT(G), вместо нее используется выражение, указанное в операторе DEF FNT в строке 40; при этом в выражение подставляется текущее значение G. При вычислении функции значение G не изменяется.
Переменная A, указанная в определении функции FNT (строка 40 приведенной выше программы) в скобках после имени функции, называется *формальным параметром.* Когда функция используется в строке 80, то переменная G выступает как *фактический параметр,* предоставляющий функции конкретное входное значение.

Оператор DEF FN
Общая форма записи: DEF FNx = *выражение или*
DEF FNx (y) = выражение
где x - имя функции, обычно ограничиваемое одной буквой; y — простая переменная, называемая параметром, или список параметров, в котором переменные указаны через запятую. Параметры локальны по отношению к определению функции (разъяснение приводится ниже). Функция может быть определена для работы со строками символов или числами, и после имени функции и переменных при необходимости указываются соответствующие признаки типа значения {$, %, #).
Примечание. Некоторые системы, например Sinclair ZX81, *не обеспечивают возможность определения пользователем функций; другие системы ограничивают применение функций, допуская в их определении только числовые значения; во многих системах, допускающих длинные имена переменных, например в системах* Microsoft *и* BBC, *допускаются и длинные имена функций; полное определение функции должно умещаться в одной строке; по поводу блочных функций см. подразд. 5.1.2.*
Приведем примеры однострочных функций:

```
DEF FNP            =3.14159
DEF FNA (S)        = S*S+4
```

```
DEF FNP (Q)         =3*B + Z*Q
DEF FNZ(Q)          =SQR(13.7*Q + A)
DEF FNA (A,B,C)   = (A + B + C)/3
```

Любое имя переменной, появившееся при определении функции в списке параметров, является локальным по отношению к этой функции. Это означает, что данная переменная отличается от переменной с тем же именем, указанной в любом другом месте программы. Она принимает значение, передаваемое ей во время обращения к функции. Другие переменные, входящие в состав "выражения", ничем не отличаются от обычных переменных программы, и во время обращения к функции используется их текущее значение. Приведенные ниже примеры демонстрируют эти свойства:

```
10 REM ПРИМЕР ОПЕРАТОРА FN
20 A=2
30 B=30
40 C=-21
50 REM
60 DEF FNX(A)=A*B+C
70 REM
80 PRINT "A =";A; B =";B; "C =";C
90 PRINT "ЗНАЧЕНИЕ 6*B+C =";FNX(6)
100 PRINT "A=";A;"B=";B;"C=";C
110 END
141
```

Исполнение строки 60 не вызывает каких-либо действий, помимо регистрации определения функции, используемой в строке 90 со значением 6. Переменная А в строке 60 является локальной (фиктивной) и обязана отличаться от переменной А, не входящей в определение функции. При распечатывании значения последней переменной до и после обращения к переменной должно изображаться одно и то же значение.

В состав указанного в определении функции выражения могут входить другие функции при условии, что все эти функции пользователя уже были определены. Иначе говоря, в него могут входить либо встроенные функции, либо функции, появившиеся в программе ранее. Например, функция

```
20   DEF FNR(X) =INT(X * RND(l) + 1)
```

возвращает значение случайного целого числа в диапазоне 1 . . . X, так что при исполнении цикла

```
30   FORI=1 TO 10
40      PRINT FNR (6)
50   NEXT I
```

будет напечатано десять случайных значений в диапазоне 1 ... 6.

Подобные однострочные функции (оператор-функции) не очень-то способны обеспечить требуемую для структурного программирования модульность, но довольно полезны, особенно в ситуации, когда в программе много раз встречается одно и то же длинное выражение.

5.1.2. БЛОЧНЫЕ ФУНКЦИИ

Применение в качестве функции не одного, а нескольких операторов гораздо ближе к общепринятому понятию функции по сравнению с однострочными функциями, допускаемыми в большинстве систем с Бейсиком. Во многих других языках программирования тело функции может быть образовано группой операторов, но лишь немногие из расширенных систем с Бейсиком обладают такой "продвинутой" возможностью.

В пределах блоков, образующих функции, можно определять дополнительные локальные переменные, похожие на параметры-переменные тем, что они совершенно не связаны с одноименными внешними переменными.

Система с Бейсиком для ЭВМ ICL 2903/4 допускает включение в образующий функцию блок операторов любого типа; блок завершается новым оператором FNEND. Определение локальных переменных помещается в строке с оператором DEF FNx после скобки, завершающей список параметров. Приведенный ниже пример иллюстрирует блочную функцию, вычисляющую N!≡N факториал, где
N!≡N*(N-1)*(N-2)* ...*3*2*1 так что
3! =3 * 2* 1 5!=5*4*3*2*1


```
10 REM ПРИМЕР БЛОЧНОЙ ФУНКЦИИ
20 DEF FNF(N) T, I
30 T=I
40 FOR I=1 TO N
50  T=T*I
60 NEXT I
70 FNF=T
80 FNEND
90 REM
100 PRINT "3! =";FNF(3)
110 PRINT "2! =";FNF(2)
120 PRINT "5! =";FNF(5)
130 END
RUN
3!=6
2!=2
5!=120
END AT LINE 130
```

Образующий функцию блок простирается от строки 20 до строки 80 (включительно) , а T и I являются дополнительными локальными переменными.
В Бейсике ICL 2903/4 можно определять рекурсивные функции. Рекурсия в данном случае означает, что функция может вызывать сама себя. Это свойство языка программирования полезно, но ценность его часто преувеличивается. Конструирование рекурсивных функций требует большого мастерства, но при исполнении программы может оказаться, что их применение неэффективно и вызывает затруднения. Приведенную выше программу вычисления факториала можно написать в виде рекурсивной функции:

```
10 REM ПРИМЕР РЕКУРСИВНОЙ ФУНКЦИИ
20 DEF FNF(N)
30 IF N=1 THEN 60
40   FNF=N*FNF(N-1)
50   GOTO 70
60 FNF=1
70 FNEND
```

В этом примере значения локальной переменной N образуют стек. Каждый раз, когда функция вызывается из самой себя, на верх стека помещается новое значение. Вызов FNF(3) приводит к тому, что N полагается равным 3 и управление передается функции FNF. При ее исполнении в строке 40 происходит вызов FNF(2), в результате чего значение 3 помещается в стек. Вызов FNF(2) приводит к тому, что N полагается равным 2, и при исполнении строки 40 при данном вызове значение 2 помещается в стек значений N и происходит вызов FNF(1),в результате которого только всего и

происходит, что возвращается значение FNF, равное 1. Теперь можно по очереди исполнить все предыдущие вызовы: каждый раз снимается верхний элемент стека значений N и умножается на текущее значение FNF. Последовательность возвратов результатов такова:
FNF=1 влечет за собой FNF(1) =1,
затем 2*FNF (1) =2*1 влечет за собой FNF (2) = 2,
затем 3*FNF (2) =3*2 влечет за собой FNF (3) = 6

Конечный результат — правильное значение 3!.
В Бейсике ВВС можно определять функцию несколькими строками и задавать локальные переменные с помощью оператора LOCAL. Определение функции завершается присваиванием (=), задающим значение функции, но не имеющим левой части. Как показывают следующие два примера, при этом соглашении в Бейсике ВВС однострочные функции имеют обычный вид:
20 DEFFNF(X) =X*X+3*X-20 . 10 DEF FNS (X) = SQRT (X+B-C)
Если определение функции занимает несколько строк, то знак присваивания (=), задающий значение функции в Бейсике ВВС, может встречаться в нем один или несколько раз. Например, следующая функция возвращает в качестве результата наибольшее из двух значений:
100 DEF FNZ(Q,R)
110 IF Q>R THEN = Q ELSE = R;
Таким образом, при исполнении оператора 10 PRINT FNZ (2,3)
будет получен результат 3. Повторим пример функции для вычисления факториала:
```
10 REM ПРИМЕР ФУНКЦИИ В БЕЙСИКЕ ВВС
20 DEF FNFACT(N)
30 LOCAL T,I
40 T=1
50 FOR I=1 TO N
60 T=T*I
7B NEXT I
80 =T
```
Знак присваивания в строке 80 служит признаком завершения определения функции и дает значение функции FNFACT ( ) .
В Бейсике ВВС функции могут быть рекурсивными, поэтому в случае применения признака завершения определения (=) можно переписать и рекурсивный пример вычисления факториала, воспользовавшись к тому же возможность дать функции более полное имя. Употребляя расширенный вариант оператора IF Бейсика ВВС, можно написать следующее определение:
```
10 REM ПРИМЕР РЕКУРСИВНОЙ ФУНКЦИИ В БЕЙСИКЕ ВВС
20 DEF FNFACTORIAL(N)
30 IF N=1 THEN =1 ELSE =N*FNFACTORIAL(N-1)
40 REM
```
Здесь определение функции занимает лишь строки 20 и 30. Учтите, что в Бейсике ВВС определения функций и процедур не должны входить в основную часть программы.
При любом применении функции следует избегать побочных эффектов. Функции должны давать один результат, передаваемый через имя функции. Все другие изменения называются *побочными эффектами;* сюда относятся

изменения значений общедоступных переменных и некоторые изменения в других объектах, например изменение положения в файле или ввод-вывод данных. Если такие эффекты необходимы, то вместо функции следует применить подпрограмму.

5.2. ПОДПРОГРАММЫ

В то время как функции возвращают только одно значение через имя функции, подпрограммы способны возвращать любое число значений. Они обладают возможностью использовать и модифицировать любую переменную или любой массив, которые доступны программе. В отличие от функций, подпрограммы не могут иметь локальных переменных, списка параметров или фиктивных переменных.

При таком подходе существует значительная опасность нечаянных и нежелательных изменений значений переменных. Многие другие языки программирования накладывают строгие ограничения на использование общедоступных (глобальных) переменных в подпрограммах и обеспечивают полную независимость между локальными переменными, используемыми в подпрограммах, и переменными, используемыми в других частях программы. К сожалению, в Бейсике этого не делается, из-за чего кое-кто считает, что Бейсик не заслуживает репутации хорошего языка программирования общего назначения. Однако в некоторых расширенных версиях Бейсика предлагаются работающие должным образом блочные процедуры (детали см. в гл. 6).

При соблюдении разумной осторожности подпрограммами можно широко пользоваться при работе по методу пошаговой детализации. Однако в задачу языка программирования входит как можно большее устранение рутинной работы программиста. И дополнительные меры предосторожности не потребовались бы, если бы подпрограммы Бейсика стандартно обладали достаточно удобными свойствами.

5.2.1. ОПЕРАТОРЫ GOSUB И RETURN

Подпрограмма представляет собой набор операторов программы, к которому можно обратиться с помощью оператора GOSUB из любой строки программы. Когда операторы подпрограммы исполнены и достигнут оператор RETURN, управление автоматически передается обратно, к оператору в строке, непосредственно следующей за GOSUB. На рис. 5.1 строки с 500-й по 600-ю представляют собой подпрограмму, которая первый раз вызывается из строки 100; после исполнения подпрограммы управление возвращается к строке 110. Вслед за этим подпрограмма вызывается из строки 200, и после ее исполнения управление возвращается к строке 210. Обратите внимание на оператор STOP в конце основной программы: если бы он отсутствовал, управление передавалось бы строке 500 и исполнение программы продолжалось бы до строки 600, при попытке исполнения которой была бы зафиксирована ошибка с выдачей соответствующего сообщения, например RETURN STATEMENT FOUND WITHOUT A GOSUB (обнаружен оператор RETURN без предшествующего GOSUB).

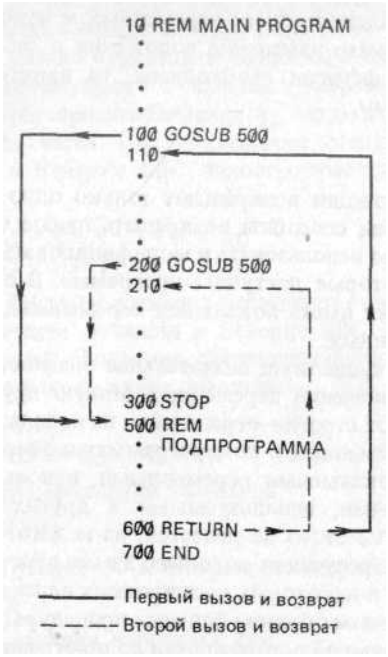


Рис. 5.1. Процесс передачи управления при двух вызовах подпрограммы

Работа с подпрограммами обеспечивается операторами GOSUB и RETURN, указываемыми в разных местах программы. Для обозначения начала подпрограммы специального оператора нет, поэтому обязательно отмечайте начало подпрограммы с помощью комментария в операторе REM. Служебное слово GOSUB, за которым следует номер оператора, вызывает передачу управления подпрограмме, начинающейся оператором с этим номером. Оператор RETURN, являющийся последним оператором подпрограммы, вызывает передачу управления обратно, к оператору, непосредственно следующему за оператором GOSUB, вызвавшим переход к подпрограмме.

В действительности, эффект применения подпрограммы тот же, как если бы каждый оператор GOSUB был заменен на совокупность всех операторов подпрограммы. Рассмотрим несколько следующих операторов:

210 FOR L=1 TO N 220 PRINT"-"; 230 NEXT L 240 PRINT

Они обеспечивают печать строки, состоящей из N знаков минуса (-), и используются в качестве подпрограммы в следующей программе:

10 КЕМ ОСНОВНАЯ ПРОГРАММА
20 PRINT "ДЛИННЫЙ ЗАГОЛОВОК"

30 N=17
40 GOSUB 200
5B PRINT "ЕЩЕ РАЗ"
60 N=7
70 GOSUB 200
88 FOR I=1 TO 5
90 N=I
100 GOSUB 200
110 NEXT I
120 STOP
200 REM ПОДПРОГРАММА ДЛЯ ПОДЧЕРКИВАНИЯ N СИМВОЛОВ
210 FOR L=1 TO N

```
220   PRINT "-";
230 NEXT L
240 PRINT
250 RETURN
260 END
RUN
ДЛИННЫЙ ЗАГОЛОВОК
----------
ЕЩЕ" РАЗ
END AT LINE 120
```

Операторы с 200-го по 250-й изображают строку из N символов и используются в данном примере для подчеркивания текста, выдаваемого операторами в строках 20 и 50. Обратите внимание на то, что значение N устанавливается перед GOSUB и доступно (наряду со значениями всех других переменных) в подпрограмме. Треугольная фигура формируется обращениями к подпрограмме из цикла FOR в строке 80. Так как L используется и изменяется в подпрограмме, то этой переменной нельзя пользоваться как управляющей в цикле FOR в строке 80.

Может оказаться полезной приведенная ниже подпрограмма. Она способна напечатать N повторений символа, имеющего код ASCII, равный M:

```
200 REM ПОДПРОГРАММА ПЕЧАТИ
210 REM N ПОВТОРЕНИЙ
220 REM СИМВОЛА С КОДОМ ASCII M
230 FOR L=1 TO N
240   PRINT CHR$(M);
250 NEXT L
260 PRINT
270 RETURN
```

Операторы GOSUB и RETURN

Общая форма записи: GOSUB s

где s - *номер оператора. Общая форма записи:*

RETURN

Управление передается подпрограмме, начинающейся с оператора, имеющего номер s. *Затем операторы выполняются по порядку до тех пор, пока не встретится оператор* RETURN, *После этого управление передается строке, следующей непосредственно за оператором* GOSUB, *вызвавшим подпрограмму.*

В некоторых версиях Бейсика предусмотрен оператор выбора одной подпрограммы из совокупности подпрограмм, аналогичный оператору ON-GOTO, обсуждавшемуся в подразд. 4.3.1, в котором GOTO заменяется на GOSUB. Любой оператор RETURN возвращает управления обратно к строке, следующей непосредственно за ON-GOSUB.

При применении подпрограмм Бейсика придерживайтесь следующих правил:

(а) Четко обозначайте начало и конец каждой подпрограммы. Если это возможно, выделяйте их путем отступа операторов вправо.

(б) Никогда не пользуйтесь оператором GOTO для входа в подпрограмму и для выхода из нее. Каждую подпрограмму надо рассматривать как независимый логически завершенный модуль.

(в) Имейте под рукой список глобальных входных переменных и выходных результатов каждой подпрограммы. Лучше всего оформить его комментариями к программе с помощью операторов REM.

(г) Обязательно убедитесь, что в подпрограмме не изменяются значения таких внешних переменных, как счетчики циклов. Если это возможно, в каждой подпрограмме вводите свой принцип именования внутренних переменных.
(д) В отличие от функций, подпрограммы не появляются в программе до того, как ими будут пользоваться, поэтому полезно группировать подпрограммы вместе после оператора STOP, указанного в конце основной программы.

5.3. РАЗРАБОТКА ПРОГРАММЫ

Хотя существует очень много различных методов разработки, ни один из них не может обеспечить решение широкого круга задач, встречающихся при обработке данных. Однако среди них выделяются два достаточно общих взаимно дополняющих метода: метод пошаговой детализации, уже представленный выше, и метод разработки структур данных.
Метод разработки структур данных был предложен М. Джексоном и другими авторами. Он очень полезен в тех приложениях обработки данных, где сравниваются и модифицируются большие файлы данных, например в системах обработки счетов-фактур, системах начисления заработной платы, расчета с покупателями, обработки каталогов изделий. Эти системы характеризуются относительно несложной обработкой (простые арифметические и логические операции) большого числа данных. Данные доминируют в задаче и метод разработки должен определять структуру программы исходя из структуры данных. К сожалению, описание методов распознавания структур данных и использования их для создания структур программ требует

гораздо больше места, чем мы располагаем в этой книге. Заинтересованные программисты могут обратится к прекрасно излагающей данный предмет книге M. A. Jackson *Principles of Program Design,* выпущенной издательством Academic Press в 1975 году.

5.3.1. МЕТОД ПОШАГОВОЙ ДЕТАЛИЗАЦИИ

Существует множество слегка отличающихся друг от друга форм этого метода и множество названий: детализация сверху вниз, последовательная конкретизация, методическое программирование и даже структурное программирование, хотя последнее название применяется ко всему процессу разработки и производства программы.
Метод пошаговой детализации используется для решения огромного множества задач и преобразования решений задач в программы для ЭВМ. Но так как в его основу положена разработка процессов, применяемых для обработки данных, то его использование обычно наиболее эффективно в тех ситуациях, когда процессы доминируют и вводимые-выводимые данные относительно просто организованы. Можно провести параллель между этапами пошаговой детализации и составлением структограмм: в то время как структограммы служат превосходным средством графического представления процессов, на них совершенно не отражены структуры вводимых-выводимых данных. Для решения тех задач, в которых доминируют структуры данных, можно пытаться применить упомянутый выше метод разработки структур данных.
В методе пошаговой детализации можно выделить следующие существенные этапы:
(а) На уровне 1 создается общее описание программы в целом. Определяются основные логические шаги, требуемые для решения задачи, даже если пока не известно, как их выполнить. Эти логические шаги могут отражать различные физические шаги решения или могут быть удобными групповыми именами для тех действий, выполнение которых представляется довольно смутно. Последовательности шагов, требуемых для решения задачи, записываются на обычном языке или на псевдокоде (см. подразд. 5.3.3) .
(б) На уровне 2 в общих терминах детализируется описание шагов, введенных на этапе (а). В детализированное описание может входить обозначение циклических структур, в то время как

действия внутри циклов могут по-прежнему оставаться неясными. Таким образом, выполняются только общие эскизы сложных действий.
(в) На уровне 3 и последующих уровнях в виде последовательных итераций производятся те же действия, что описаны на этапе (б). При каждой новой итерации уточняются детали, остававшиеся неясными после предыдущих итераций, и создаются более определенные описания. По мере выполнения итераций неопределенные детали становятся все проще и проще, так что на каком-то этапе могут быть полностью описаны.
(г) Разработка завершена: в модульном виде получено описание требуемой программы. Перевод этого описания в программу на Бейсике должен быть достаточно простой задачей.

(д) Если в процессе пошаговой детализации возникли детали и переменные, не зависящие от остальных частей программы, или произошло дублирование модулей, то подобные модули являются очень вероятными кандидата ми в подпрограммы или функции. В подобной ситуации модуль описания используется для создания тела процедуры или функции и заменяется на вызов процедуры или функции.
Чтобы в результате разработки получилось описание, пригодное для перевода в удовлетворительную программу, лучше всего при применении метода пошаговой детализации пользоваться только определенными программными структурами. Последние детально обсуждаются в подразд. 5.3.2, а в оставшейся части настоящего подраздела приводится простой пример, демонстрирующий применение описанного выше процесса.
Пусть требуется написать программу, которая получает в качестве ввода значение N, равное числу учеников в классе, а также рост, возраст и вес каждого ученика. Эта программа должна выдать средние значения всех трех показателей. Вначале применим этап (а) ко всей задаче в целом:
Уровень 1
Ввести число учеников. Вычислить средний рост. Вычислить средний возраст. Вычислить средний вес.
Теперь применим этап (б) к каждому из описанных выше шагов:
Детализация 1.1. Вычислить средний рост Ввести рост каждого ученика. Вычислить с помощью цикла средний рост. Изобразить результаты.
Детализации 1.2 и 1.3 аналогичны приведенной выше, но применяются к возрасту и весу вместо роста. В результате получен
Уровень 2
ВВОД "Число учеников"; N.
(1.1) для роста
(1.2) для возраста
(1.3) для веса
содержащий ссылки на каждую детализацию. Продолжим этот процесс и, поскольку все три детализации практически одинаковы, рассмотрим детализацию 1.1. Раскроем суть трех действий, составляющих ее описание.
Детализация 2.1. Ввести рост каждого ученика. ПЕЧАТЬ приглашения к вводу значений. ВВОД значений в массив с помощью цикла или оператора MAT INPUT.
Детализация 2.2 Вычислить с помощью цикла средний рост.
Организовать цикл, пробегая N значений и накапливая сумму в S. Поделить S на N для получения среднего роста.

Детализация 2.3. Изобразить результаты.
ПЕЧАТЬ названия показателя (например, "ВОЗРАСТ") и среднего значения.
Так как производится по сути одно и то же расширение всех трех основных частей уровня 2, то представляется естественным реализовывать указанные выше действия в виде подпрограмм общего назначения, передавая им в строковой переменной название показателя - рост, возраст, вес. Таким образом, уровень 3 примет следующий вид:
Уровень 3:
ВВОД "Число учеников"; N
Присвоить переменной значение "РОСТА".
Вызвать детализации (2.1), (2.2) и (2.3).
Присвоить переменной значение "ВОЗРАСТА".
Вызвать детализации (2.1), (2.2) и (2.3).
Присвоить переменной значение "ВЕСА".
Вызвать детализации (2.1), (2.2) и (2.3).
КОНЕЦ
Проявив некоторое старание, мы получили настоящую модульную структуру. При этом неявным образом было принято решение вводить значения всех показателей в один и тот же массив, скажем A ( ) , и передавать название показателя через одну строковую переменную, скажем A$. С учетом сказанного детализации можно закодировать в виде подпрограмм следующим образом:

```
(2.1)  500 REM ПОДПРОГРАММА ВВОДА ЗНАЧЕНИЙ
510 PRINT 'ВВЕДИТЕ";N;"ЗНАЧЕНИЙ ";A$
520 FOR I=1 TO N
530       INPUT A(I).
540   NEXT I
550   RETURN
(2.2)  400   REM ПОДПРОГРАММА ВЫЧИСЛЕНИЯ СРЕДНЕГО
410 S=0
420   FOR I=1 TO N
430   S=S+A(I)
440   NEXT I
450   S=S/N
460  RETURN
(2.3) 700  REM ПОДПРОГРАММА ВЫВОДА ЗНАЧЕНИЙ
710 PRINT "СРЕДНЕЕ ЗНАЧЕНИЕ ";A$;" РАВНО";S
720 RETURN
```

Так как все три действия всегда выполняются последовательно, то можно сэкономить на повторяющихся вызовах, поместив их также в подпрограмму. Таким образом, получается следующая подпрограмма процесса:

```
200   REM ПОДПРОГРАММА ПРОЦЕССА
210  GOSUB 500
220  GOSUB 400
230  GOSUB 700
240  RETURN
```


Обратите внимание на то, что эта подпрограмма по существу представляет собой детализацию 1.1. Основная программа получается как перевод в операторы Бейсика уровня 3:

```
10 REM ОСНОВНАЯ ПРОГРАММА 20 DIM A(100)
30 INPUT "ЧИСЛО УЧЕНИКОВ";N 40 A$="РОСТА" 50 GOSUB 200 60 A$="ВОЗРАСТА"
```

70 GOSUB 200 ,
80 A$="ВЕСА" 90 GOSUB 200 100 STOP
Может потребоваться указание после текста подпрограмм заключительного оператора 750 END. На каждом этапе процесса детализации описываемые действия воплощались в виде модулей, большинство из которых в конечном счете становились подпрограммами. Сделать это в данном простом примере было легко, но в принципе модули могли оставаться группами операторов в подпрограмме или в основной программе.
На рис. 5.2 показана результирующая структура программы. Каждый оператор GOSUB обозначен горизонтальной линией со стрелкой, ведущей к соответствующей подпрограмме. Для простоты картины операторы в каждом прямоугольнике опущены.
Если этот рисунок положить на бок так, что прямоугольник с основной программой окажется наверху, то будет видно, что основным свойством данного метода разработки является то, что структура наращивается сверху вниз. Более ранние идеи состояли в том, что вначале надо формировать базис

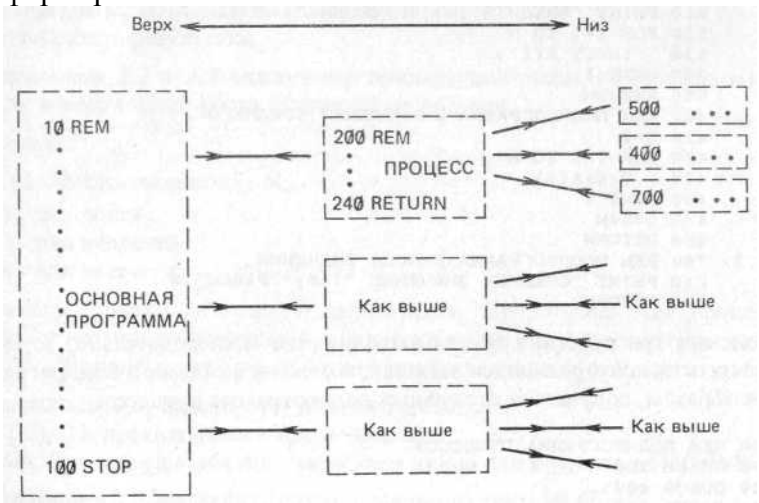


Рис. 5.2. Структура вызовов модулей и возвратов управления для примера программы из подразд. 5.3.1

Рис. 5.3.

для модулей высоких уровней, так как модули нижних уровней выполняют простые специфичные действия. А при описанном выше методе Вы можете планировать проведение определенных расширений структур программы прежде, чем узнаете в точности, как действуют модули нижнего уровня. Однако при этом Вам потребуется принять некоторые решения, которые повлияют на структуру модулей нижних уровней, а Вам необходимо знать, можно ли будет эти модули преобразовать в программу на Бейсике для Вашей ЭВМ. Например, нет никакого смысла предполагать, что эти модули могут использовать матричные (MAT) функции, если таковые в Вашей системе не предусмотрены. Поэтому применение метода детализации "сверху вниз" требует умения прогнозировать выполнимость принимаемых решений.

Каждый модуль имеет вид, приведенный на рис. 5.3. Таким образом, задача разработки программы сводится к составлению спецификаций модулей. По крайней мере теоретически, по составленной спецификации каждый модуль можно написать и отладить независимо от других. Это не вполне справедливо для Бейсика ввиду глобального использования переменных и опасности случайного изменения важных значений в подпрограмме.

Необходимо подчеркнуть, что приведенный выше пример программы был подобран так, чтобы она распадалась на простые модули, которые могли быть преобразованы в подпрограммы. В других задачах простого дублирования действий на каждом этапе может и не быть и при разработке из решения модули могут войти в основную программу.

Покажем теперь, как данный метод разработки можно применять, используя структограммы вместо употребленной выше смеси текста на обычном языке с переводами служебных слов Бейсика.

Действуйте следующим образом:

(а) Для изображения уровня 1 нарисуйте большой прямоугольник на всю страницу. Он должен изображать программу в целом. Начните заполнять его шагами, требуемыми для решения задачи. Изображения шагов, как правило, будут прямоугольниками с описанием действий общего вида, однако иногда может потребоваться символ цикла.

(б) Для изображения уровня 2 нарисуйте отдельно каждый из прямоугольников действий, полученных в (а), и заполните их некоторыми действиями в виде символов внутри прямоугольника.

(в) Для изображения уровня 3 и последующих уровней продолжайте разработку каждого прямоугольника с изображением процесса до тех пор, пока детали не будут проработаны до конца. По мере проработки деталей символы вложенных итераций и принятия решения становятся все меньше и меньше, однако не забывайте, что любой прямоугольный символ можно выделить из диаграммы и развивать как отдельный модуль, т. е. как подпрограмму или функцию.

(г) Изучение разработки в целом, независимо от того, умещается она в прямоугольнике на одной странице или распределена по нескольким прямо-

угольникам, приводит к законченной схеме написания команд программы на Бейсике. Замените подходящие прямоугольные блоки на подпрограммы или функции, используя критерии, обсуждавшиеся на этапе (д) предыдущей спецификации разработки.

Применение структограмм рассчитано на то, чтобы Вы ограничивались употреблением правильных программных структур, так как при использовании чисто словесных описаний легче отклониться от следования принципам структурного программирования. Первый этап разработки в терминах структограмм примет следующий вид: .

Ввести число учеников
(1.1) Вычислить средний рост
(1.2) Вычислить средний возраст
(1.3) Вычислить средний вес
Конец

Числа в прямоугольниках идентифицируют детализации. Если взять общий процесс "вычислить средний показатель", то его можно детализировать следующим образом:

(1.1)

(2.1) Ввести N значений
(2.2) Цикл для вычисления среднего
(2.3) Распечатать результаты

Конечно, при желании на этом этапе можно было бы добавить больше деталей, например ввести символы повторения операций для (2.1) и (2.2). Детализация отдельных указанных выше процессов приводит к следующим структограммам:

(2.3)

ПЕЧАТЬ результатов
ВОЗВРАТ

Большинство из этих деталей можно было бы набросать в большом исходном прямоугольнике, отложив выбор границ подпрограмм до более поздних этапов разработки. Не забывайте, что любой прямоугольный символ является потенциальным кандидатом для преобразования в подпрограмму или функцию.

5.3.2. ДЕТАЛИЗИРОВАННЫЕ СТРУКТУРЫ

Беспорядочному использованию операторов Бейсика для конструирования программы следует предпочесть метод, целью которого является достижение правильности, ясности и простоты. Добиться этого нетрудно, если ограничиваться группами операторов Бейсика, удовлетворяющими этим принципам. При разработке программ с использованием структограмм это происходит автоматически. Так как соответствующие символам структограмм операторы в Бейсике чаще всего не предусмотрены, требуется определенная аккуратность, чтобы образовывать эти структуры из более примитивных операторов, таких, например, как GOTO.

Ниже описаны три типа структур, каждый из которых иллюстрируется структограммами и фрагментами программ, написанных на стандартном Бейсике (минимальном подмножестве) и на расширенной версии Бейсика.

Последовательность, или *конкатенация,* операторов представляет собой такой блок операторов, в котором нет передачи управления ни внутрь этого блока, ни из него. Управление всегда передается первому оператору блока и покидает блок лишь после исполнения его последнего оператора. Таким образом, такой блок имеет только один вход и один выход:

```
Процесс
10   REM БЛОК ПРОЦЕССА
20----------
30----------
40----------
50----------
60   REM КОНЕЦ БЛОКА
```

В некоторых системах с Бейсиком несколько операторов можно записать в одной строке, используя разделители; при условии, что среди этих операторов нет оператора передачи управления, такую строку можно рассматривать как небольшой блок.

Выбор обеспечивает исполнение одной из двух альтернатив, каждая из которых может быть представлена единственным оператором или последовательностью операторов. Выбор лучше всего представить следующим образом:

"Если условие выполнено, то процесс 1, в противном случае процесс 2"

Приведем пример реализации выбора на Бейсике:

```
10 IF Y=l THEN 100
28 X=2                    -----Процесс 2
за L=X+6      ----/
40 GOTO 150 l00l
X=l                       -----Процесс 1
на L=Y-X      ---/
150 ---------
```

Этот пример выглядит довольно запутанным. Он мог бы быть яснее, если бы процессы были представлены подпрограммами:

```
10   IF Y = 1 THEN GOSUB 500
20   IF NOT (Y = 1) THEN GOSUB 400
```

Этот пример не слишком элегантен, но, по крайней мере, достаточно хорошо следует форме структуры выбора. Во многих версиях Бейсика предусмотрен расширенный оператор IF (см. гл. 6), которым при возможности следует пользоваться:

```
10   IF Y = 1 THEN X = 1: L = Y-X ELSE X = 2: L = X+6
```

Правда, обычно этот оператор ограничен одной строкой, поэтому последовательность операторов каждой альтернативы записывается через двоеточия. Конечно, иногда процесс 2 оказывается пустым и мы имеет выбор

"Если условие выполнено, то процесс 1" примеры которого таковы:

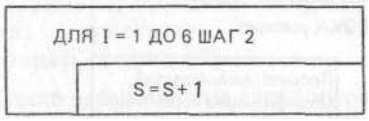
```
10   IF Y= 1 THEN X= 10 или
10   IF Y=1 THEN GOSUB 200
```

В данном случае можно уменьшить занимаемое структограммой место, если деформировать ее так, чтобы пустая часть альтернативы не занимала много места. Например, вместо

можно использовать

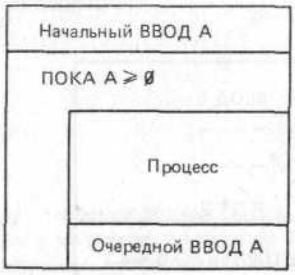
Итерация является повторением последовательности операторов, включающим некоторую форму цикла. Основная форма итерации "пока условие выполнено, исполнять процесс" подразумевает проверку условия в начале каждого прохода цикла. В стандарте Бейсика подобный оператор общего вида отсутствует, но зато в нем предусмотрены операторы FOR-NEXT, представляющие собой особый случай цикла "пока .. . , исполнять ...", в котором условие гласит, что значение управляющей переменной лежит в допустимых пределах. Например,

```
10   FOR I =1 TO 6 STEP 2
20      S = S+1 30   NEXT I
```



Если же условие имеет сложный вид, то стандартный Бейсик ничем не может помочь и цикл надо конструировать с помощью операторов IF и GOTO. Помните, что проверка выполнения условия составляет первую часть цикла. В следующем примере цикл требуется для повторения ввода данных до тех пор, пока не будет введено отрицательное число:

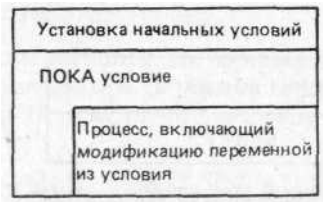
```
        10 INPUT A
      --> 20 IF A<0 THEN 100      ----------
      |   30 ------                         |
      |   40 ------                         |
Цикл  |   50 ------                         |
      |   60   INPUT A                      V  Выход из цикла
      <-- 70 GOTO 20
         100 ------
```



Обратите внимание на то, что условие в строке 20 является отрицанием условия цикла ПОКА.
Чтобы устранить дублирование оператора INPUT A, приведенную выше структограмму можно запрограммировать следующим образом:

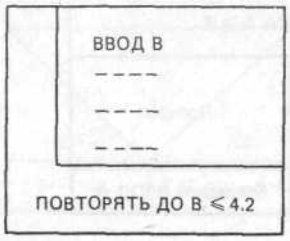
```
      --> 10 INPUT A
      |   20 IF A<0 THEN 100      ----------
      |   30 ------                         |
      |   40 ------                         |
Цикл  |   50 ------                         |
      |   60 ------                         V  Выход из цикла
      <-- 70 GOTO 10
         100 ------
```

Однако в этом случае концом цикла будет не оператор в строке 80, а оператор в строке 10. Строка 10 служит также инициирующим оператором перед входом в цикл, и цикл "переворачивается с ног на голову" для того, чтобы включить в себя этот оператор. Хотя этот прием экономит оператор, данная форма цикла трудна для понимания и поэтому не рекомендуется. Таким образом, общая форма обсуждаемого цикла такова:



Только в нескольких версиях Бейсика имеется оператор, организующий цикл ПОКА (см. гл. 6).
Другой вариант итерации возникает в том случае, когда проверка условия происходит в конце цикла. Он имеет форму "повторять процесс до тех пор, пока не выполнится условие". В данной ситуации стандартный Бейсик вообще ничем не может помочь, поэтому для конструирования такого цикла надо пользоваться операторами IF и GO TO, например:

```
10    INPUT B
20    -------
30    -------
40    -------
50  IF B > 4.2 THEN 10
```



Обратите внимание на то, что условие в структограмме является отрицанием условия в операторе IF. Структограммы разработаны с таким расчетом, чтобы горизонтальная полоса символа цикла указывала место проверки условия окончания цикла.
Лишь немногие версии Бейсика предусматривают конструкцию ПОВТОРЯТЬ ДО (см. оператор REPEAT в гл. 6).

5.3.3. ПСЕВДОКОД

Псевдокод фактически представляет собой словесный аналог структограмм. Он включает в себя наборы фраз для написания обсуждавшихся выше групп операторов (последовательность, выбор, итерация), дополняемых текстом на обычном языке. Псевдокод не имеет строгого определения; сконструируйте свой псевдокод, используя следующие элементы:
Комментарии Формулы
Описания действий и процессов Последовательности
Выбор — "если .. . , то .. ., в противном случае ..." Итерации — "пока . .., повторять ..." — "повторять ... до ..."
В описания процессов могут входить такие операторы Бейсика, как INPUT, PRINT, READ и присваивания[1], но не операторы перехода или другие средства передачи управления, применение которых должно ограничиваться реализацией трех указанных выше типов структур на заключительном этапе процесса проектирования.
Концепцию псевдокода легче всего понять на примере. Пусть требуется прочитать серию данных, определить наибольшее значение и вывести эти данные, поделенные на наибольшее значение. Скажем, если данные представляют собой последовательность чисел
4,2.51,10.0,-5.0,7.5 то вывод должен выглядеть следующим образом:

0.40,0.251, 1.00,-0.5,0.75 Первый уровень разработки ясен:
Уровень 1: Ввести данные. Найти максимум. Вывести результаты.
[1] Так как по замыслу псевдокод должен быть в достаточной мере близок к естественному языку, то далее в контексте псевдокода будут использоваться переводы служебных слов Бейсика, набранные прописными буквами. - *Прим. перев.*

Детализация 1.1. Ввод данных можно детализировать на псевдокоде следующим образом: Получить число элементов данных. Пока не все элементы введены прочитать и запомнить значение элемента. Конец цикла
Это описание можно перевести на Бейсик следующим образом:

```
100   INPUT "УКАЖИТЕ ЧИСЛО ЭЛЕМЕНТОВ"; N
110   FOR I=1 TO N 120      INPUT A (I);
130   NEXT I
```

Детализация 1.2. Отыскание максимума можно детализировать следующим образом:
Выбрать в качестве максимума элемент данных.
Пробежать все введенные значения, заменяя текущий максимум на очередное значение, если оно превысило его.
Детализация 1.3. Вывод результатов можно детализировать следующим образом:
Пока не все результаты выведены,
ВЫВОД (значение очередного элемента)/максимум. Конец цикла.
Это описание можно немедленно перевести на Бейсик:

```
300   FOR I = 1 TO N
310     PRINT A(I)/M
320   NEXT I
```

Уровень 2
Он включает в себя три детализированные выше части, из которых только (1.2) требует дополнительного внимания. Ее можно детализировать на псевдокоде следующим образом:
Положить M = первый элемент данных. Пока не все элементы просмотрены,
если M < текущий элемент, то M = текущий элемент. Конец цикла.
Это описание можно перевести на Бейсик следующим образом:

```
200  M=A(1)
210   FOR I=1 TO N
220     IF M < A (I) THEN M = A (I)
230   NEXT I
```

Так как приведенные выше модули используются только по одному разу и очень просты, то можно не делать из них подпрограммы, а объединить их вместе в одну программу:


```
10   REM ПРИМЕР ПРОГРАММЫ, ПОЛУЧЕННОЙ ИЗ ПСЕВДОКОДА
20   DIM A(20)
100   INPUT "ВВЕДИТЕ ЧИСЛО ЭЛЕМЕНТОВ";N
110   FOR I=1 TO N 120   INPUT A(I),
130   NEXT I
200   M=A(1)
210   FOR I=1 TO N
```

```
220   IF M<A(I) THEN M=A(I)
230   NEXT I
300   FOR I=1 TO N
310   PRINT A(I)/M;
320   NEXT I
500   END
RUN
ВВЕДИТЕ ЧИСЛО ЭЛЕМЕНТОВ ?5
?4.2.51.10.0.-5.0.7.5
0.40  0.251  1.00 -0.5  0.75 END AT LINE 500
```

При решении реальной задачи может потребоваться написать на псевдокоде много уровней, чтобы довести все модули до такого состояния, при котором они окажутся готовыми для программирования.

5.3.4. ЗАКЛЮЧЕНИЕ

(а) Разработка по методу "сверху вниз". Решение задачи представляется в виде модулей, каждый из которых выполняет самостоятельную группу действий. Если на начальных этапах Вы обнаруживаете, что не можете продолжать разработку далее, то причиной этому могли быть неверные исходные предпосылки. Может быть, Вы не вполне поняли задачу. Однако попытки получить решение нередко являются составной частью процесса изучения задачи. Напишите решение заново, используя другой подход, например выберите иные управляющие структуры и другой порядок выполнения действий.

Иногда бывает трудно приступить к решению некоторых задач; в таких случаях может оказаться полезным взять большой лист бумаги, например чистую бумагу для распечаток, и заполнить его группами различных действий, описаниями свойств решения, формулами и возможными программными структурами, скажем массивами, — в общем, всем, что хоть в какой-то мере может оказаться полезным. Не сдавайтесь, и постепенно Вам удастся связать некоторые идеи по поводу решения в единое целое и начать формальный процесс разработки.

(б) Детализируйте действия каждого модуля либо словесно, например, на псевдокоде, либо графически в виде структограммы. Одни программисты предпочитают словесный способ изложения мыслей, другие — графический. Если применение структограмм кажется затруднительным, то причиной этому может быть жесткий контроль над теми управляющими структурами, которыми программист может воспользоваться в программе. В любом случае избегайте произвольных передач управления в программе.

(в) В заключение переведите разработанное описание из псевдокода или структограммы в программу на Бейсике. На этом этапе можно пользоваться

и оператором GO TO, но не иначе как для реализации правильных управляющих структур. Старайтесь использовать для переменных и функций содержательные имена. Включайте в программу достаточное число операторов REM с описанием ее действий и стремитесь к аккуратному расположению текста программы. Выделенные в процессе разработки модули могут стать подпрограммами или функциями либо блоками в другой подпрограмме, функции или основной программе.

(г) Сохраните наряду с распечаткой программы бумаги с набросками проекта и, если потребуется, — его заключительный вариант. Таким образом Вы получите некоторую документацию к завершенной программе.

Избегайте соблазна создавать программу по частям, не отходя от ВТУ: набрать несколько операторов на клавиатуре, исполнить их, исправить ошибки, перезапустить, добавить еще несколько операторов и т. д. Этот способ хорош, пока надо проверить, как работают те или иные приемы программи-

рования, но через какое-то время отойдите от машины и спокойно разработайте программу на бумаге.

5.4. ПРИМЕР РАЗРАБОТКИ

Деятельность в области искусственного интеллекта насчитывает много попыток использования ЭВМ для написания связного текста, рассказов и даже новелл. До сих пор они оказывались несостоятельными, тем не менее ниже приводится короткая программа, которая пытается связать несколько сотен слов в рассказ. Результаты оказываются довольно любопытными, причем способ конструирования предложений гарантирует их грамматическую правильность[1].

Основу для рассказов составляют случайным образом выбираемые образцы предложений, содержащие случайным образом выбранные слова. Образцы предложений образованы строками ключевых слов, представляющих собой название частей речи. При составлении рассказа эти ключевые слова замещаются обычными словами. Приведем пример образца предложения:

"PERSON VT POSSESSIVE ADJ NOUN"

Обычные слова одного и того же типа указываются в одной строке, начинающейся с соответствующего ключевого слова, например:

"NOUN: JAGUAR, ROVER 3500, COAT, CAT, TROUSERS"

Кроме ключевых слов в образцах предложений могут присутствовать и любые обычные слова, а также знаки препинания, например:

"PERHAPS PERSON WILL PREP THE NOUN"

[1] Так как грамматические правила русского и английского языков существенно отличаются, то простая замена английских слов, из которых составляется рассказ, на их русские эквиваленты ни к чему хорошему не приводит. Поэтому в приведенной ниже программе переведены только комментарии. - *Прим. перев.*

В программе используются следующие части речи:

Существительные	NOUN
. Притяжательные местоимения	POSSESSIVE
Личные местоимения	PERSON
Переходные глаголы (прошедшее время)	VT
Непереходные глаголы(настоящее время)	VI
Предлоги	PREP
Прилагательные	ADJ
Наречия	ADV

Общий образ действий программы таков: выбирается образец предложения и просматривается в поисках ключевых слов NOUN, POSSESSIVE, PERSON и т. д. Если обнаружено ключевое слово, то берется соответствующий список обычных слов и одно из них извлекается случайным образом для замещения ключевого слова.

При разработке предполагается выделить как можно больше самостоятельных модулей, что представляется идеальным решением для таких систем, которые предоставляют возможность написания блочных процедур, например BBC. В действительности, программа, послужившая прообразом данной разработки, была всецело скомпонована из блочных функций, но так как они не очень широко распространены, то разработка была изменена так, чтобы в результате получились подпрограммы-блоки.

Уровень 1

Инициировать требуемые переменные и функции.
Заполнить соответствующие массивы строк символов образцами
предложений и списками слов.

Выбрать N — число предложений, которые надо составить.
Образовать и напечатать N предложений.
Вся работа выполняется в последнем из приведенных выше элементов. Его можно выполнить в виде подпрограммы.
Детализация 1.1. Образовать и напечатать N предложений. Пока требуются новые предложения,
выбрать случайный образец предложения,
образовать предложение в соответствии с выбранным образцом,
напечатать полученное предложение. Конец цикла.
Уровень 2
Описание уровня 2 образуется описанием уровня 1 и детализацией 1.1. Выберем основной объект, требующий дальнейшего развития и находящийся в детализации 1.1, и развернем его в виде подпрограммы.
Детализация2.1. Образовать предложение в соответствии с выбранным
образцом
Пока еще остались списки слов (правила), просмотреть предложение и заменить в нем

каждое вхождение ключевого слова на одно из обычных слов списка. Конец цикла.
Уровень 3
Описание уровня 3 образуется из описания уровня 2 и детализации 2.1. Оно достаточно ясно, и только операция "просмотреть предложение ..." требует расшифровки. Развернем эту операцию в подпрограмму.
Детализация 3.1. Просмотреть предложение и заменить в нем каждое вхождение ключевого слова на одно из обычных слов списка Извлечь тип слова из начала списка слов (правила) . Найти число вхождений этого ключевого слова (типа слова)
в предложение.
Пока еще остались вхождения,
извлечь случайным образом обычное слово из списка, заменить вхождение на это слово. Конец цикла.
Уровень 4
Большинство из приведенных выше операций достаточно детализировано, и для завершения описания уровня 4 надо уточнить разве лишь операцию "извлечь случайным образом слово из списка".
Детализация 4.1. Извлечь случайным образом слово из списка. Найти число обычных слов в списке (правиле) . Получить случайное число. Извлечь по этому числу соответствующее слово.
Уровень 5
Таким образом, мы достигли уровня 5. На этом этапе большинство из описанных действий находится достаточно близко к окончательным программным операторам. Это обеспечивает непосредственный перевод в операторы Бейсика после одной дополнительной итерации, на которой надо принять определенные конкретные решения о требуемых строковых функциях, об именах и об использовании важных переменных.
Перед выполнением этой работы надо включить в описание одну очень полезную модификацию. Возвратимся к детализации 1.1; в ней после образования предложения дается инструкция "напечатать полученное предложение", которая должна превратиться в простой оператор PRINT. Однако в результате столь безыскусной печати слова в конце строк экрана ВТУ могут обрываться посередине и заканчиваться на следующих строках. В принципе это может оказаться приемлемым, но для придания выводу хорошего вида вместо этого оператора надо использовать подпрограмму.
Детализация 1.2. Напечатать полученное предложение.

Используя строковую переменную в качестве буфера, добавить предложение к текущему содержимому буфера.
164

Если число символов в буфере меньше длины строки ВТУ, то выйти из подпрограммы,
в противном случае напечатать часть строки вплоть до конца подходящего слова и удалить напечатанные символы из буфера.
Теперь уже в уровень 2 войдут две детализации: 1.1 и 1.2. Этот новый вариант все еще требует определенной доработки.
Детализация 2.2. Напечатать полученное предложение.
Добавить предложение к текущему содержимому буфера.
Если число символов в буфере < длина строки, то выйти из подпрограммы,
в противном случае положить "длина" = длина строки.
Пока в позиции "длина" находится символ, не являющийся пробелом, уменьшать "длина" на 1.
Конец цикла.
Напечатать символы из буфера в количестве "длина".
Удалить из буфера напечатанные символы, а также пробелы, оказавшиеся после этого ведущими.
Перед тем как приступить к описанию уровня 6, вернемся к уровню 1 и переработаем его в операторы основной программы. При этом для большинства систем потребуется генератор случайных чисел в диапазоне 1 ... N.
Уровень 6
Определить функцию, генерирующую случайные числа в диапазоне 1 ... N.
Определить необходимые массивы строк символов.
Занести образцы предложений в массив S$( )
Занести списки слов (правила) в массив R$( ) .
Получить случайное число N в диапазоне 15 ... 25 для использования в
качестве числа предложений в абзаце. Вызвать детализацию (1.1) для получения N предложений. Закончить исполнение.
(Подпрограмма, отвечающая детализации (1.1)) Занести в буфер B$( ) пустую строку. Для J от 1 до N выбрать в качестве образца предложения
S$(случайное число) и занести его в T$,
вызвать детализацию (2.1) для преобразования T$ в настоящее
предложение,
вызвать детализацию (2.2) для вывода предложения T$ в хорошем виде. Следующий проход цикла. Возврат.
(Подпрограмма, отвечающая детализации (2.2)) Для I от 1 до R (до числа правил) положить B1 $ равным правилу S$ (I),
вызвать детализацию (3.1) для замены каждого ключевого слова обычным словом.
165

Следующий проход цикла Возврат
(Подпрограмма, отвечающая детализации (3.1))
Найти положение (:) в B1$.
Переписать в WS ключевое слово правила, находящееся слева от (:).
Переписать в S$ список обычных слов, находящийся справа от (:)
Найти число O вхождений ключевого слова W$ в предложение T$.
Для I1 от 1 до 0 вызвать детализацию (4.1), возвращающую случайное слово в B1$.

Следующий проход цикла Возврат
(Подпрограмма, отвечающая детализации (4.1))
Занести в О число слов в списке S$,
подсчитать число запятых в этом списке. Получить случайное число в диапазоне 1 ... О. Найти по запятым соответствующее слово в списке S$.
Занести это слово в B1$. Возврат.
(Подпрограмма, отвечающая детализации (2.2)) Добавить предложение T$ в буфер B$. Занести в L число символов в B$. Если L < = длина строки (40) , то возврат, в противном случае положить L = 40.
Пока L-й символ строки B$ отличается от пробела,
L = L-1. Конец цикла.
Напечатать L первых символов строки B$. Пока L-й символ строки B$ является пробелом,
L = L+1. Конец цикла.
Сдвинуть содержимое буфера B$ влево, удалив L первых символов. Возврат. Конец программы.
Одна из причин, по которой приходится выполнять так много операций со строками символов, состоит в том, что элементы образцов предложений и правил имеют переменную длину, вследствие чего при извлечении и замещении элементов приходится подсчитывать занятые и пробелы и определять позиции. Можно было бы упростить программу, если бы каждый из этих элементов хранился в отдельном элементе массива строк символов, например, за счет применения двумерных массивов S$(I,J) и RS(I, J) . Однако при таком подходе становится труднее изменять образцы предложений и списки слов, и одним из достоинств разработанной выше программы является простота экспериментирования с различными списками слов и образцами предложений.

Обратите внимание на то, что подпрограмма, соответствующая детализации (2.2), может быть использована и во многих других приложениях как процедура вывода текста по словам. В действительности и вся эта программа может быть использована для других приложений, так как она просто размещает объекты по образцам. Такими объектами, например, могут быть графические элементы; в списках вместо слов можно указывать команды управления перемещением курсора, обеспечивающие вычерчивание каких-либо фигур. Созданные предложения послужат образцами для изображения рисунков на всем экране или на его части — машина станет художником. Аналогично, если имеется звуковой генератор, можно разработать систему для получения машинной музыки. Используя строковые функции Бейсика Microsoft, уровень 6 можно перевести в следующую законченную программу:

```
10 REM СОЧИНЕНИЕ КОМПЬЮТЕРА
20 REM ФУНКЦИЯ FNA ГЕНЕРИРУЕТ СЛУЧАЙНОЕ ЧИСЛО
22 REM В ДИАПАЗОНЕ ОТ 1 ДО N
30 DEF FNA(N) = INT(N*RND+1)
40 REM
58 DIM S$(5) ,R$(10) ,P1(20)
60 S=5
70 S$(1)="PERSON VT POSSESSIVE ADJ NOUN."
80 S$(2)="PERHAPS PERSON WILL VI PREP THE NOUN."
90 S$(3)="ADV. PERSON VT THE NOUN."
100 S$(4)="PREP THE ADJ NOUN PERSON VT."
110 S$(5)="VI THAT, PERSON WILL ADV VI THE ADJ NOUN."
120 REM
130 R=8
140 R$(1)="NOUN:BLACK HOLE,STARSHIP,PLANET,MOON,SUN"
```

```
150 R$(2)="POSSESSIVE:HIS,HER,THEIR"
160 R$(3)=;"PERSON: THE GALACTIC LORD,THE DARK ONE,HE,SHE"
170 R$(4)="VT:HIT,TURNED,WENT.FLEW,STOPPED,LEFT"
180 R$(5)="VI:LOOK,STOP,LEAVE,NOTICE,RUN,DESTROY"
190 R$(6)="PREP:UNDER,THROUGH,ROUND,ON,NEAR,CLOSE TO"
200 S (7)="ADJ:BRIGHT,PULSING,LONELY,FEEBLE"
210 R$(8)="ADV:SLOWLY,QUIETLY,SWIFTLY,SUDDENLY"
220 REM
230   N=15+FNA(10)
240   GOSUB 270
250  STOP
260   REM
270   REM ПОДПРОГРАММА ВЫВОДА N ПРЕДЛОЖЕНИЙ
280   PRINT
290   B$=""
300   FOR J=l TO N
310   T$=S$(FNA(S))
320   GOSUB 390
330   GOSUB 720 340   NEXT J
350   PRINT B$
360   PRINT
370   RETURN
380   REM
390   REM ПОДПРОГРАММА ПОДСТАНОВКИ СЛОВ В T$ ПО ПРАВИЛАМ
395   REM В R$ ).
400   REM ВХОД: T$ В КАЧЕСТВЕ ОБРАЗЦА ПРЕДЛОЖЕНИЯ.
405   REM ВЫХОД: РЕЗУЛЬТАТ ПОДСТАНОВКИ В T$.
410   FOR I=1 TO R
167
```

```
420   B1$=R$(I)
430   GOSUB 470
440 NEXT I
450 RETURN
460 REM
470 REM ПОДПРОГРАММА ЧАСТИЧНОЙ ПОДСТАНОВКИ СЛОВ В T$
475 REM СОГЛАСНО ПРЕДПИСАНИЮ, СОДЕРЖАЩЕМУСЯ В B1$
480 REM ВХОД: T$,B1$
485 REM ВЫХОД: T$ С ЧАСТИЧНО ПОДСТАВЛЕННЫМИ СЛОВАМИ
490 C=INSTR(B1$,":")
500 W$=LEFT$(B1$,C-1)
510 S$=RIGHT$(Bl$ ,LEN(B1$)-C)
520 REM W$ СОДЕРЖИТ КЛЮЧ (НАЗВАНИЕ ЧАСТИ РЕЧИ),
525 REM A S$ - СПИСОК СЛОВ
530 Y$=T$
540 XS=W$
550 GOSUB 900
560 FOR 11=1 TO 0
```

```
570   GOSUB 630
580   P=INSTR(T$,W$)
590   T$=LEFT$(T$,P-1)+B1$+RIGHT$(T$,LEN(T$)-P-LEN(W$)+1)
600 NEXT I1
610 RETURN
620 REM
630 REM ПОДПРОГРАММА СЛУЧАЙНОГО ВЫБОРА ИЗ СПИСКА СЛОВ
640 REM ВХОД S$, ВЫХОД - СЛОВО В B1$
650 Y$=","+S$+","
660 X$="."
670 GOSUB 900
680 I2=FNA(O-1)
690 B1$=MID$(Y$,P1(12)+2,P1(I2+1)-P1(12)-2)
700 RETURN
710 REM
720 REM ПОДПРОГРАММА КРАСИВОЙ ВЫДАЧИ ТЕКСТА
730 REM ДОБАВЛЯЕТ ПРЕДЛОЖЕНИЕ T$ В БУФЕР B$
740 REM ДЛЯ ПЕЧАТИ ПРЕДЛОЖЕНИЯ В НЕСКОЛЬКО СТРОК
745 REM НЕ РАЗРЫВАЯ СЛОВА
750 REM ВХОД: ЗАКОНЧЕННОЕ ПРЕДЛОЖЕНИЕ В T$
760 B$=B$+" "+T$
770 L=LEN(B$)+1
780 IF L< = 41 THEN RETURN
790 L=41
800 IF MID$(B$,L,1>=" " THEN 830
810   L=L-1
820   GOTO 800
830 PRINT LEFT$(B$, L-1)
840 IF MID$(B$,L,1)<>" " THEN 870
850   L=L+1
860   GOTO 840
870 B$=RIGHT$(BS,LEN(B$)-L+1)
880 RETURN
890 REM
900 REM ПОДПРОГРАММА ОПРЕДЕЛЕНИЯ ЧИСЛА ВХОЖДЕНИЙ ДЛЯ
905 REM БЕЙСИКА MICROSOFT
910 REM ВХОД: СТРОКА Y$, ОБРАЗЕЦ В X$ ДЛЯ ПОИСКА В Y$
920 REM ВЫХОД: ЧИСЛО ВХОЖДЕНИЙ В O
930 REM МАССИВ P1( ) СОДЕРЖИТ ПОЗИЦИИ ВХОЖДЕНИЙ
940 O=0
950 I3=1
96B J3=LEN(Y$)
970 K3=LEN(X$)
980 M3=0
990 P3=INSTR(I3,Y$.X$)
168


1000 IF P3<>0 THEN 1030
```

```
1010   M3=l
1020   GOTO 1070
1030 0=0+1
1040 P1(O)=P3
1050 I3=P3+K3
1060 IF(I3+K3)>J3 THEN M3=l
1070 IF M3<>1 THEN 990
1080 RETURN
1090 END
```

Последняя подпрограмма, образованная строками с 900-й по 1080-ю представляет собой модификацию программы, описанной в подразд. 4.2.5, и добавлена здесь ввиду того, что в версии Бейсика Microsoft нет стандартной функции, возвращающей число вхождений вырезки X$ в строку Y$. Позиции вхождений регистрируются в массиве P1 ( ) для дальнейшего использования. Вызовы этой процедуры обеспечиваются строками 530 . . . 550 и 650 .. . 670. В некоторых системах предусмотрены функции с большими возможностями, возвращающие за одно обращение к ним как число вхождений, так и позиции этих вхождений, поэтому окончательные детали реализации уровня 6 зависят от того, какими стандартными функциями располагает Ваша система.

Возвратимся к изначальной цели программы. В результате ее исполнения по команде RUN Вы получите забавный, но не вполне связный рассказ. Причина этого проста: ЭВМ хранит правила грамматики и манипулирует словами, но в основу рассказа должны быть положены идеи, которыми либо можно, либо нельзя манипулировать случайным образом. Слова являются лишь внешним проявлением глубинных идей. Если бы ЭВМ могла манипулировать идеями, а не словами, то тогда можно было бы получать поразительные результаты.

УПРАЖНЕНИЯ

5.1. Разработайте и напишите программу для перевода температуры из шкалы Цельсия в шкалу Фаренгейта, определив функцию, выполняющую это преобразование. Исходите из того, что признаком конца списка вводимых значений температуры по шкале Цельсия служит нуль.

5.2. Разработайте и напишите программу, выдающую письма стандартной формы, адресуемые многим людям. От письма к письму должны меняться только фамилия и адрес, поэтому для печати писем примените подпрограмму, использующую переменную информацию из основной программы.

Письма подобного вида часто посылаются фирмами, сообщающими сведения о своей продукции, или лоббистскими группами. обращающимися ко всем членам правительства с просьбой проголосовать за принятие определенного законопроекта.

Программа должна выдавать ряд писем на основе информации, запомненной в операторах DATA. Если у Вас нет принтера, то выводите результаты обычным путем на ВТУ.

5.3. Разработайте и напишите программу, выдающую список лиц, упорядоченный по возрасту (от младшего к старшему). Сведения о каждом лице должны иметь формат

ФИО, возраст (например, И. В. СЕМЕНОВ, 24) .

Придумайте десять лиц и занесите сведения о них, не упорядочивая каким-либо способом, в операторы DATA. Для вывода и сортировки примените подпрограммы.

ЧАСТЬ III (МАТЕРИАЛ ПОВЫШЕННОЙ ТРУДНОСТИ)

Этот материал рассчитан на опытных программистов. В нем в дополнение к материалу повышенной трудности по программированию обсуждаются некоторые специальные темы.

6. РАСШИРЕНИЯ БЕЙСИКА

В этой главе описываются некоторые дополнительные средства, которыми обладают многие версии Бейсика. В ней наряду с вполне стандартными средствами, например оператором PRINT USING и логическими операторами, обсуждаются расширения Бейсика, например управляющие структуры типа циклов REPEAT (ПОВТОРЯТЬ ДО), различные типы переменных и средства получения псевдографических изображений.

Некоторые расширения Бейсика превратились в самостоятельные языки, например COMAL, который будет вкратце описан, чтобы показать возможное направление развития Бейсика в будущем.

6.1. ВЫРАЖЕНИЯ

В подразд. 1.2.5 детально обсуждались требования, предъявляемые к соглашениям о порядке вычисления арифметических выражений; в данном разделе обсуждаются все типы операций и их упорядочение по старшинству.

На рис. 6.1. показан относительный приоритет всех операций, доступных в большинстве версий Бейсика. Наивысшее старшинство имеют скобки, поэтому во вложенных выражениях в первую очередь вычисляются те части выражений, которые заключены в скобки на самом внутреннем уровне. Таким образом, при вычислении выражения

D + X*(C + X*(B+X))

надо выполнить только два умножения и результат в точности тот же, что и при вычислении выражения

D+C*X+B*X*X+X*X*X

Если одно и то же старшинство имеет более чем одна операция, то они выполняются слева направо. Например, на рис. 6.1 показано, что умножение и деление имеют одно и то же старшинство, поэтому

A/B*C равно $\frac{A \cdot C}{B}$

но

A/(B*C) равно $\frac{A}{B \cdot C}$

Отрицание представляет собой операцию, изменяющую знак содержащегося в переменной значения. Если A содержит 37.21, то —A будет равно —37.21. Следовало бы всегда давать возможность записи

B=C*-A

	Стандартные операторы	Дополнительные операторы
Высокий приоритет		
Скобки	( )	(Только в Бейсике BBC: - NOT)
Возведение в степень	↑ или **	(Только в Бейсике BBC:↑)
Изменение знака	-	
Умножение, деление	• /	
Деление нацело		(Только в Бейсике BBC: MOD DIV) (Только в Бейсике Microsoft: \ MOD)
Сложение,	= <>	

вычитание		
Операции отношения	<>	
	<=>=	
	NOT	
	AND	
	OR	
	XOR	(В Бейсике BBC используется EOR)
	IMP	(В некоторых системах)
	EQV	(В некоторых системах)
Низкий приоритет		

Рис. 6.1. Перечень операций в порядке старшинства. Вначале выполняются операции с наивысшим приоритетом; если соседние операции имеют равный приоритет то они выполняются по порядку слева направо

так как отрицание по старшинству выше умножения, но некоторые системы отвергают такую запись и требуют форму

B = C*(-A)

которую легче понять как программисту, так и системе.

В некоторых системах предусмотрена операция целочисленного деления, обозначаемая обратной косой чертой (\) или словом DIV. Операцию можно указывать как между целыми, так и между вещественными переменными (см. разд. 6.2) , но перед выполнением деления оба операнда округляются до целых чисел, и для большинства микроЭВМ результат округления должен находиться в допустимом диапазоне —32 768 . . . +32 767. Целочисленное деление выполняется несколько быстрее по сравнению с делением вещественных чисел. К примеру,

17\4 дает 4 17 DIV 4 дает 4

и

3.142\2 дает 1

Функция MOD также имеет дело с целочисленным делением, но возвращает целое значение, являющееся остатком от такого деления. Таким образом,

17 MOD 4 дает 1

Как DIV, так и MOD можно применять к вещественным значениям, и системы округляют их до ближайших целых чисел перед выполнением этих операций:

3.142\1.7дает 1

и

3.142 MOD 1.7 дает 1

поскольку вычисляются как 3/2 (после округления), что дает результат 1 и остаток 1.

В некоторых системах (например, в Бейсике BBC) вместо округления производится отбрасывание дробной части; при таком подходе

3.142 DIV 1.7 дает 3 и

3.142 MOD 1.7 дает 0 что получается в результате вычисления 3/1.

Операции отношения по старшинству одинаковы. Они младше арифметических операций, действуют между парами арифметических выражений и не могут указываться последовательно:

A > = B правильно

A > = B = C неправильно

Для конструирования сложных выражений, подобных последнему из приведенных выше, надо применять логические операторы, обсуждаемые в следующем разделе. Операции отношения введены для формирования логического результата по значениям двух арифметических выражений. Этот логический результат — ИСТИНА или ЛОЖЬ — используется в условных (или булевских) выражениях всюду, где это требуется в Бейсике, например в операторе

IF A > = B + C THEN 200

или в расширенной управляющей структуре вида REPEAT

UNTIL A > = B

Общее правило для всех систем таково: "ИСТИНА"≠0 "ЛОЖЬ"=0

Как же значения ИСТИНА и ЛОЖЬ представляются в ЭВМ? В Бейсике они представляются в виде двоичных целых значений, так что

Нулевому значению эквивалентна ЛОЖЬ Ненулевому значению эквивалента ИСТИНА

Система	Результаты операций отношения	
	ИСТИНА	ЛОЖЬ
Sinclair ZX81	1	0
ICL 2904	1	0
Microsoft	–1	0
PET	–1	0
BBC	–1	0

Таблица 6.1. ***Значения, используемые логическими операциями и операциями отношения***

Так как операции отношения образуют логические выражения, то они должны производить указанные выше значения для значений ИСТИНА и ЛОЖЬ. Вообще говоря, так и делается, но чаще всего вместо случайного ненулевого значения для представления значения ИСТИНА в зависимости от системы выбирается +1 или -1. В табл. 6.1 приведены эти значения для некоторых систем. Таким образом, если в системе ИСТИНА представляется числом -1,то

PRINT (5 > 3) изобразит -1 PRINT (5 > 7) изобразит 0

и в результате выполнения оператора A = (5 > = 2)

переменной A будет присвоено значение -1. Наличие скобок не обязательно, однако в некоторых системах их отсутствие считается ошибкой. Можно пользоваться и переменными, как, например, в выражении

C = (A > B)

и сложными выражениями, например C= ((A>B) AND(Z< = Y))

Такое употребление операций отношения затрудняет понимание и в общем случае не рекомендуется, однако может оказаться полезным там, где требуется образовывать двоичные комбинации, используя целые переменные, или, как это сделано выше, переходить к обработке значений, получаемых из логических выражений.

6.1.1. ЛОГИЧЕСКИЕ ОПЕРАЦИИ

К логическим операциям относятся операции NOT (НЕ), AND (И), OR (ИЛИ). Эти операции надо применять в логических выражениях, при вычислении которых получаются значения либо ИСТИНА, либо ЛОЖЬ. В результате применения этих операций также получаются результаты либо ИСТИНА, либо ЛОЖЬ. Например, в основу выражения

IF A < B AND B > = C THEN 500

положено сравнение двух логических значений, находящихся по обе стороны от операции AND. Смысл таких выражений легко понимать, читая их как фразы, написанные на английском языке, но ради полноты изложения ниже
173

приводится таблица, в которой И соответствует значению ИСТИНА, и Л ЛОЖЬ:

Условное выражение 1	Условное выражение 2	Условное выражение 1 AND условное выражение 2	Условное выражение 1 OR условное выражение 2
И	И	И	И
И	Л	Л	И
Л	И	Л	И
Л	Л	Л	Л

Операция NOT по старшинству выше остальных логических операций. Она меняет значение ИСТИНА на ЛОЖЬ и наоборот. Обратите внимание на то, что операция AND дает значение ИСТИНА только в том случае, если оба условных выражения имеют значение ИСТИНА, а операция OR дает значение ИСТИНА во всех случаях, кроме ситуации, когда оба условных выражения имеют значение ЛОЖЬ. Приведенные ниже соотношения представляют собой часть правил логического вывода; в них через a, b и c обозначены логические значения (ИСТИНА или ЛОЖЬ) :
(a AND b) AND c = a AND (b AND c) (a OR b) OR c = a OR (b OR c) a AND b = b AND a a OR b =b OR a
и
a OR (b AND c) = (a OR b) AND (a OR c) a AND (b OR c) = (a AND b) OR (a AND c)
в то время как действие NOT подчиняется правилам
NOT (a AND b) = (NOT a) OR (NOT b) NOT (a OR b) = (NOT a) AND (NOT b)
Последние правила очень полезны на практике для упрощения трудно понимаемых выражений — например, в место
IF NOT (A > B AND B < = C) THEN 500 можно написать
IF (A <= B OR B>C) THEN 500
Чтобы иметь полный набор символических логических манипуляций, в некоторых системах с Бейсиком к описанным выше операциям добавляются еще три: XOR (в Бейсике BBC используется запись EOR), IMP и EQV, действие которых показано в приведенной ниже таблице:
174

Условное выражение 1	Условное выражение 2	Условное выражение 1 XOR условное выражение 2	Условное выражение 1 IMP условное выражение 2	Условное выражение 1 EQV условное выражение 2
И	И	Л	И	И
И	Л	И	Л	Л
Л	И	И	И	Л

Л	Л	Л	И	И

В результате выполнения стандартной операции OR будет получено значение ИСТИНА, если оно имеет первый операнд, либо второй, либо и первый, и второй. А исключающая операция XOR (exclusive OR — исключающее ИЛИ) означает "либо", и ее применение к операндам, которые одновременно имеют значение ИСТИНА, приводит к результату ЛОЖЬ. Эта операция используется чаще других логических операций. Если она отсутствует, то ее можно сконструировать из стандартных операций с помощью соотношения
a XOR b = (a OR b) AND (NOT (a AND b))
IMP является сокращением от implication (импликация) . Этим термином именуется логическое следствие, т. е. конструкция "если a, то b". Изучение приведенной выше таблицы истинности показывает, что эта конструкция приводит к результату ЛОЖЬ только в том случае, если из истинной посылки a вытекает ложный вывод b.
EQV представляет собой сокращение от equivalence (равенство) и является двусторонней импликацией, означающей "a, если, и только если b". Эту операцию можно получить из простой импликации с помощью соотношения
a EQV b=(a IMP b) AND (b IMP a)

6.1.2. МАНИПУЛИРОВАНИЕ БИТАМИ

Логические операции действуют точно таким же образом, как и операции отношения; при этом программа явным образом пользуется значениями ИСТИНА и ЛОЖЬ. Приведенный ниже фрагмент программы иллюстрирует такое употребление этих значений для мини-ЭВМ, в которой ИСТИНА представлено значением 1 (микроЭВМ может действовать иначе, см. ниже) :

```
10  REM ДЕМОНСТРАЦИЯ ДЕЙСТВИЯ ЛОГИЧЕСКИХ ОПЕРАЦИЙ
15  REM (НЕ ДЛЯ МИКРО-ЭВМ)
20  INPUT "ВВЕДИТЕ A,B";A,B
30  PRINT "NOT A РАВНО" ; (NOT(A) )
40  PRINT "NOT B РАВНО";(NOT(B))
50  PRINT "A AND B =";((A)AND(B))
60  PRINT "A OR B =" ;( (A)OR(B))
70  STOP
80  END
```



```
RUN
ВВЕДИТЕ A,B
?4.0
NOT A РАВНО 0
NOT B РАВНО 1
A AND B=0
A OR B = 1
END AT LINE 80
RUN
ВВЕДИТЕ A,B
?12.13
NOT A РАВНО 0
NOT B РАВНО 0
```

```
A AND B = 1
A OR B = 1
END AT LINE 80
RUN
ВВЕДИТЕ A, B
?-67.90
NOT A РАВНО 0
NOT B РАВНО 0
A AND B = 1
A OR B = 1
END AT LINE 80
```

Результаты исполнения этой программы согласуются с принятым в Бейсике общим соглашением, что ЛОЖЬ представляется нулевым значением, а ИСТИНА - ненулевым. В данном случае, в системе ICL 2904, результаты выполнения операции производились из принятого в этой системе выбора представления ИСТИНЫ (в нашем случае +1), а не из произвольного ненулевого значения (см. табл. 6.1) . Допустима запись

```
IF (B) THEN 500
```

где B содержит значение, которое может быть интерпретировано как ИСТИНА или ЛОЖЬ. Но остерегайтесь беспорядочного применения этой конструкции, так как оператор

```
IF (A) OR(B>C) THEN 500
```

означает вовсе не то, что можно себе представить, читая его как предложение на английском языке.

Результаты применения логических операций к произвольным значениям у многих микроЭВМ несколько различаются. Учтите, однако, что если значениям ЛОЖЬ и ИСТИНА соответствуют 0 и -1 (или +1 в некоторых системах) , то результатами применения логических операций должны быть эти же числа. Если в качестве операндов задать произвольные значения, то логические операции будут действовать побитовым образом. Иначе говоря, они возьмут соответствующие биты каждого из операндов и выполнят над ними требуемую операцию, что даст один бит результата. Стандартная таблица фактически дублирует те, что были приведены ранее, но показывает поведение отдельных битов (только 0 или 1) при выполнении над ними логических операций:

Бит a	Бит b	Бит a AND бит b	Бит a OR бит b	Бит a XOR бит b	Бит a IMP бит b	Бит a EQV бит b
1	1	1	1	0	1	1
1	0	0	1	1	0	0
0	1	0	1	1	1	0
0	0	0	0	0	1	1

Прежде чем выполнять операции над битами, большинство систем для микроЭВМ преобразует операнды в 16-битовые целые числа в диапазоне —32 768 ... +32 767, представляемые в дополнительном коде (в виде дополнения до двух) . Если хотя бы один из операндов лежит вне этого диапазона, то фиксируется ошибка.

Следовательно, исполнение предыдущей программы на большинстве микроЭВМ приведет к следующим результатам:

```
10 REM ДЕМОНСТРАЦИЯ ДЕЙСТВИЯ ЛОГИЧЕСКИХ ОПЕРАЦИЙ
15 REM (ДЛЯ МИКРО-ЭВМ)
20 INPUT "ВВЕДИТЕ A,B";A,B
```

```
30 PRINT "NOT A PABHO";(NOT(A))
40 PRINT "NOT B PABHO";(NOT(B))
50 PRINT "A AND B=";((A)AND(B))
60 PRINT "A OR B =";((A)OR(B))
70 END
RUN
ВВЕДИТЕ A.B?63.16
NOT A PABHO -64
NOT B PABHO -17
A AND B = 16
A OR B=63
END AT LINE 70
RUN
ВВЕДИТЕ A.B?-1.6
NOT A PABHO 0
NOT B PABHO -9
A AND B = 8
A OR B = -1
END AT LINE 70
RUN
ВВЕДИТЕ A.B?-1.-2
NOT A PABHO 0
NOT B PABHO 1
A AND B = -2
A OR B = -1
END AT LINE 70
```

Чтобы придать им смысл, используйте приведенную выше таблицу побитовых операций и не забудьте, что числа представляют в дополнительном коде.

Например,

63 равно		0000	0000	0011	1111
—64 равно		1111	1111	1100	0000
16 равно		0000	0000	0001	0000
-17	равно	1111	1111	1110	1111
-1	равно	1111	1111	1111	1111
0	равен	0000	0000	0000	0000
8	равно	0000	0000	0000	1000
-9	равно	1111	1111	1111	0111
-2	равно	1111	1111	1111	1110

						0
1	равно	0000	0000	0000	000	
						1

Двоичные представления всех этих чисел даны в 16-битовом дополнительном коде. Для удобства чтения биты сгруппированы по четыре. Логическими операциями можно пользоваться для образования или "маскировки" определенных битов в байте, которым можно манипулировать в программе на Бейсике с помощью операторов PEEK (вытащить) и POKE (поместить). Кроме того, эти операции могут понадобиться при исполнении операций ввода-вывода, например для проверки значений различных битов байта состояния, описывающего действие дискового контроллера.

6.2. ТИПЫ ПЕРЕМЕННЫХ

Во всех версиях Бейсика предусмотрен тип переменных, предназначенный для хранения вещественных чисел. Переменные этого типа используются и для запоминания целых чисел: так, скажем, число 127 можно запомнить в вещественной переменной как 127.0. Основной проблемой применения любого типа переменных является точность представления чисел, так как в ЭВМ для каждой переменной выделяется ограниченный объем памяти. Чем больше битов выделяется для переменной, тем выше точность представления чисел. Однако при выполнении любой арифметической операции над вещественными числами возникают ошибки округления, которые в конечном итоге могут оказаться вполне ощутимыми. Их влияние может проявляться даже в тривиальных случаях; например, если A = 1 и B = 2, то вполне может оказаться, что A*B равно не 2, а 1.999999, так что проверка, равно ли A*B числу 2, даст отрицательный результат! В действительности описанная выше ситуация может и не возникнуть, так как при выполнении арифметических операций автоматически предпринимаются определенные действия по округлению результата. Один из способов избежать проблем, вызываемых ошибками округления, состоит в использовании целых переменных во всех ситуациях, где в особенности требуются точные значения, например при управлении циклами и при проверках выполнения условий. В некоторых версиях Бейсика для этих целей предусмотрены специальные целые переменные (табл. 6.2), для обозначения которых после имени переменной указывается знак %. Таким образом,

Таблица 6.2. *Наличие различных типов переменных*

Система	Целые переменные	Вещественные переменные	
		с одинарной точностью	с двойной точностью
Microsoft	Да	Да (7 значащих цифр)	Да (16 значащих цифр)
BBC	Да	Да (от 9 до 10 значащих цифр)	Нет
CBM PET	Да	Да (9 значащих цифр)	Нет
Sinclair	Нет	Да (от 9 до 10 значащих цифр)	Нет
ICL 2904	Нет	Да (от 11 до 12 значащих цифр)	Нет

A%
TOP%
COUNTERS%
являются допустимыми именами целых переменных.
На случай, если обычной точности вещественных переменных оказывается недостаточно, в некоторых системах предусмотрены вещественные переменные с двойной точностью, у которых число значащих цифр примерно вдвое больше. Для их обозначения используется знак #. Так,
DIVISOR#
T#
SUM#
являются допустимыми именами переменных с двойной точностью. Конечно, такие переменные требуют почти вдвое больше места в памяти ЭВМ по сравнению с (вещественными) переменными с одинарной точностью. К тому же и времени на умножение или деление переменных с двойной точностью требуется гораздо больше, чем для переменных одинарной точности.
Если в системе допускаются целые переменные или переменные с двойной точностью, то в ней допускаются и массивы этих типов. Приведенное ниже описание использования целых переменных и переменных с двойной точностью применимо также и к элементам массивов соответствующих типов.

6.2.1. ЦЕЛЫЕ ПЕРЕМЕННЫЕ

Под эти переменные нередко выделяется два байта памяти, вследствие чего они могут содержать только целые числа в диапазоне ±32 767. Однако

такой диапазон вполне адекватен большинству приложений этих переменных, например применению их в качестве счетчиков циклов:

```
FOR COUNT% = 1 TO 500
```

или в качестве индексов массивов:

```
B(I%) =A(L%) +10.6
```

или в операторах выбора:

```
ON SELECT% GOSUB 100, 200, 400, 900
```

или в условных выражениях:

```
IF T% = 99 THEN GOSUB 293
```

При выполнении арифметических операций над целыми числами надо использовать специальную операцию целого деления, так она выполняется быстрее деления вещественных чисел. Эта операция, упоминавшаяся в разд. 6.1, обозначается обратной косой чертой (\) или служебным словом DIV.
Преобразование типов данных из вещественных в целые совершается автоматически при выполнении оператора присваивания. При этом результат может округляться, например:

```
10 A%=9.8
20 PRINT A% RUN
```

или у результата может отбрасываться дробная часть, например:

```
10 A%=9.8 20 PRINT A%
RUN
9
```

Прежде чем пользоваться преобразованием типов, выясните, каким именно образом Ваша система выполняет эту операцию. При переносе программ с одной системы на другую можно гарантировать одинаковое их действие только в том случае, если они сами выполняют округление; например, применение оператора

```
A%=INT(B+0.5)
```

обеспечивает преобразование положительного значения B в ближайшее большее целое число, если дробная часть B больше или равна 0.5, и в ближайшее меньшее целое число, если дробная часть B

меньше 0.5. В табл. 6.3 перечислены функции, возвращающие целый результат. Если в Вашей системе функции FIX нет, то вместо нее можно использовать выражение

SGN(X)*INT(ABS(X))

При написании программ следует пользоваться целыми переменными всюду, где только возможно, так как при этом требуется меньше памяти, арифмети-

Таблица 6.3. *Функции, возвращающие целые результаты*

Функция	Действие	Пример
INT(X)	Наибольшее целое <= X	INT(21.8) возвращает 21 INT(-21.8) возвращает -22
Только в Бейсике Microsoft		
CINT(X)	Округление X	CINT(21.8) возвращает 22
FIX(X)	Отбрасывание дробной части X	FIX (21.8) возвращает 21 FIX (-21.8) возвращает -21

ческие операции над ними выполняются быстрее, чем над вещественными числами, в толковании управляющих операторов не возникает недоразумений.

В Бейсике BBC целые переменные только на один байт короче вещественных переменных и занимают по четыре байта, что позволяет хранить значения в диапазоне $\pm 2 * 10^{12}$. При этом переменные с A% по Z% представляют собой особый случай: в них можно сохранять значения при переходе от одной программы к другой с помощью команды CHAIN; кроме того, их можно использовать при работе с присоединяемыми процедурами.

6.2.2. ПЕРЕМЕННЫЕ С ДВОЙНОЙ ТОЧНОСТЬЮ

В большинстве систем с Бейсиком вещественные числа представляются с точностью до 6—7 значащих цифр; в некоторых системах точность достигает 11—12 значащих цифр. Иногда предусматривается функция типа EPS, возвращающая наименьшее значение, для которого справедливы соотношения

1-EPS<K1+EPS

Если в Вашей системе такой функции нет, то точность стандартных вещественных переменных можно определить, запустив следующую программу:

```
10  E=1
20   E=E/2
30   IF   (1-E)<1   AND   1<(1+E)   THEN   20
40   PRINT "ТОЧНОСТЬ РАВНА";E
50   END
RUN
ТОЧНОСТЬ РАВНА 3.63798E-12
END AT LINE 50
```

Полученный результат говорит о том, что точность достигает 11—12 значащих Цифр. Если операция AND в Вашей системе отсутствует, то строку 30 можно заменить на оператор IF (1 —Б) < 1 THEN 20. Результатом исполнения этой программы на персональной ЭВМ BBC будет 1.16E-10. Если в Вашей системе предусмотрены переменные с двойной точностью, то для определения их точности приведенную выше программу надо исполнить, заменив в ней E на E#.

Переменными с двойной точностью надо пользоваться в исключительных случаях, так как арифметические операции над ними выполняются медленно, а памяти для их хранения требуется гораздо больше, чем для вещественных переменных с одинарной точностью. Применение некоторых численных методов требует использования переменных с двойной точностью, но обычно в немногих наиболее критичных частях программы. Такими частями могут быть внутренние циклы, в которых многократно повторяется вычисление арифметических выражений, или области, в" которых результаты зависят от разности больших чисел. По возможности массивы с двойной точностью следует избегать, так как при их применении время исполнения программы изменяется от нескольких секунд до нескольких минут.

При выполнении оператора присваивания значения переменной с двойной точностью происходит автоматическое преобразование типа значений, однако при применении арифметических операций могут возникать недоразумения. Так, оператор

```
D# = A/B*C
```

не обеспечит большей точности, чем оператор

```
D = A/B*C
```

так как все переменные в правой части имеют одинарную точность и, следовательно, арифметические операции будут выполняться также с одинарной точностью; только при присваивании результата (в первом примере) произойдет преобразование в значение с двойной точностью. Общее правило состоит в том, что перед вычислением арифметического выражения все операнды преобразуются в значения с одинаковой точностью. Так, при выполнении оператора

```
D# = A#/B*C
```

и арифметические операции, и результат будут иметь двойную точность; при выполнении оператора

```
D = A#/B*C
```

арифметические операции также будут выполнены с двойной точностью, но при присваивании точность результата будет уменьшена до одинарной. На самом деле это может быть вполне удовлетворительным, так как может оказаться, что источником численных ошибок является только арифметическое выражение. В некоторых системах константам можно приписывать особый признак, показывающий, что они имеют двойную точность. В этом случае 6.45#/7.12 вычисляется с двойной точностью, а 6.45/7.12 -с одинарной. В табл. 6.4 приведены две функции для явного преобразования вещественных чисел с двойной точностью в вещественные числа с одинарной точностью и наоборот. При исследовании устойчивости численных методов нередко бывает полезно иметь универсальную подпрограмму или функцию для уменьшения числа значащих цифр у значений переменных для изучения эффекта снижения точности. Это уменьшение можно реализовать двумя спосо-

Таблица 6.4. *Функции Бейсика Microsoft для преобразования значений с двойной точностью*

Функция	Действие	Пример
CDBL(X)	Преобразует значение с одинарной точностью X в значение с двойной точностью	CDBL (454.67) возвращает 454.6700134277
CSNG(Y)	Преобразует значение с двойной точностью Y в значение с одинарной точностью	CSNG (1.234567890) возвращает 1.2345678

бами. Во-первых, можно просто округлить любое значение до D-й десятичной позиции:

```
100   REM ПОДПРОГРАММА ОКРУГЛЕНИЯ X ДО D-Й
ДЕСЯТИЧНОЙ ПОЗИЦИИ
110   X = INT(X*10**D+SGN(X)*.5)/10**D
```

```
120   RETURN
```
Такое простое округление непригодно во многих случаях, когда значения сильно различаются по абсолютной величине. В этих случаях надо иметь дело со "значащими цифрами". Это означает, что независимо от значения числа надо сохранять определенное число цифр, несущих информацию о числе. Например, при округлении до трех значащих цифр
1.2345E + 5 превратится в 1.23E+5 и
1.2345E-10 превратится в 1.23E-10
Если округляемое значение масштабировать так, что оно оказывается в диапазоне 0.1 ... 1, то после этого можно применять указанный выше метод округления; таким путем приходим к программе
```
100 REM ПОДПРОГРАММА СНИЖЕНИЯ ТОЧНОСТИ X ДО
105 REM N ЗНАЧАЩИХ ЦИФР
110 IF X=0 THEN RETURN
120 S=1.0
130 IF ABS(X)<1 THEN 170
140   X=X*.1
150  S=S*10
160   GOTO 130
170 IF ABS(X)>.1 THEN 210
180  X=X*10
190   S=S*.1
200   GOTO 170
210 X=INT<X*10**N+SGN(X)*.5>*S/10**N
220 RETURN
```
Если в системе присутствует функция для вычисления $\log_{10}$, то ее можно использовать для масштабирования X вместо двух циклов в приведенной выше программе. Другой способ уменьшения точности представления значе-

ния X — преобразовать его в строку символов и обработать как последовательность символов.
Следите за тем, чтобы при уменьшении числа значащих цифр арифметические выражения обрабатывались должным образом. Например, выражение
надо заменить на следующий фрагмент:
```
50   X = B(I)-T
60   GOSUB 100
70   X = X/A(I,J)
80   GOSUB 100
90   Y(I) =X
```
6.2.3. ПРИСВАИВАНИЕ ТИПОВ ДАННЫХ
В некоторых версиях Бейсика (например, Microsoft) существует специальный оператор, объявляющий, что все переменные, имена которых начинаются с определенной группы символов, имеют заданный тип. Например, благодаря оператору
```
10   DEFINT I, K, L
```
все переменные, имена которых начинаются с I, K, L, автоматически будут иметь целый тип. Во всех версиях Бейсика по умолчанию предполагается вещественный тип, но с помощью операторов DEFINT, DEFSNG, DEFDBL и DEFSTR можно установить заданным переменным соответственно целый тип, вещественный тип с одинарной точностью, вещественный тип с двойной точностью, строковый тип. Для указания интервала букв можно использовать знак дефиса (-). Оператор
```
10   DEFDBL A-E
```
эквивалентен оператору
```
10   DEFDBL A, B, C, D, E
```

Явное указание типа знаками #, $, % и ! после имени переменной перекрывает объявление типа с помощью DEF. Например, в программе

```
10   DEFDBL A, B, C 20   B! =4.2
30   C%=6
```

переменная C% будет восприниматься как целая, а B! — как вещественная с одинарной точностью. (В версии Бейсика Microsoft знак ! можно использовать для явного указания вещественного типа с одинарной точностью) .

Чтобы избежать недоразумений, лучше всего выработать собственные соглашения по поводу выбора начальных букв для разных типов — например, оставить I для целых переменных, S — для строковых и D — для вещественных переменных с двойной точностью.

6.3. ОПЕРАТОР PRINT USING

Оператор PRINT USING позволяет придавать выводимым числам и строкам символов требуемый формат там, где они могут появиться. Хотя автоматическое форматирование вывода, обеспечиваемое стандартным оператором PRINT, является очень гибким, но существуют ситуации, когда оно оказывается неудовлетворительным: например при выводе таблиц, печатании финансовых документов, бухгалтерских балансов, чеков.

Оператор PRINT USING — один из наиболее сложных в Бейсике и в разных системах реализуется по-разному. Ниже описаны общие наиболее часто встречающиеся варианты его применения.

6.3.1. ФОРМАТЫ

Каждый оператор PRINT USING содержит либо формат, представляющий собой образ формы представления выводимых значений, чисел или строк символов, либо ссылку на такой формат. Основной составной частью такого формата является символ решетки (#), указывающий, что надо напечатать цифру или символ. Действие формата показано в приведенном ниже примере, иллюстрирующем также различия между двумя операторами PRINT:

```
10 REM ДЕМОНСТРАЦИЯ ДЕЙСТВИЯ ОПЕРАТОРА PRINT USING
2 B FOR I=1 TO 6
30   READ A
40   PRINT A
50 NEXT I
60 PRINT
70 RESTORE
80 FOR I=1 TO 6
90   READ A
100   PRINT USING "-####.## ;A 110 NEXT I
120 DATA 2.2,1000,47.345,-6.25,.037,1 130 END
RUN 2.2 1000 47.345 -6.25 3.7E-2 1
2.20
1000.60
47.35
-6.25
0.04
1 .00
END AT LINE 130
```

Последняя группа чисел аккуратно выровнена по правому краю поля, при этом числа округлены до второго знака после десятичной точки и дополнены незначащими нулями.

Образ формата представляет собой строку символов, следующую за служебным словом USING (используя). Каждый формат задается в виде строки.
185

Позиции выводимых цифр строго определены расположениями знаков #. Десятичная точка печатается, так 'как она указана спустя четыре символа от начала формата, и выводимые числа выравниваются по десятичной точке. В образе формата десятичная точка может быть указана в любом месте или вовсе отсутствовать.
В табл. 6.5 собраны общеупотребительные символы, служащие для описания формата. Пожалуй, реализации числовых форматов в достаточной мере согласованы, но для описания форматов вывода строк символов существует несколько различных систем. Вначале рассмотрим вывод чисел. В табл. 6.6 приведено несколько примеров действия различных форматов; при указании в операторе PRINT USING такие форматы заключаются в кавычки.
Таблица 6.5. *Широко используемые символы описания формата. Через* A *и* B *обозначены две различные системы описания формата вывода строк символов*

Описание формата	Действие для чисел	Действие для строк символов
#	Вывод одной цифры, или нуля, или пробела	В системе A — вывод одного символа (см. ниже комментарий)
.	Вывод десятичной точки	
,	Вывод запятой через каждые три позиции слева от десятичной точки	
$ ИЛИ £	Вывод $ или £ в указанной позиции	
$$ или ££	Плавающий вывод $ или £.Только один символ выводится непосредст-венно перед числом	
+	Может быть указан до или после #. В этой позиции будет напечатан знак числа (+ или -)	
-	То же, что и + выше, но печатается только знак —	
↑↑↑↑, или	Означает поле для вывода экспоненты	
*	Вывод *во всех неиспользуемых позициях слева от десятичной точки	

186

Таблица 6.5 (окончание)

Описание формата	Действие для чисел	Действие для строк символов
\ \		Две обратные косые черты с N пробелами между ними означают вывод N + 2 первых символов строки (Система вывода строк символов В)
!		Вывод первого символа строки (Система вывода строк символов В)
&		Вывод всей строки (Система вывода строк символов В)

Таблица 6.6. *Примеры числовых форматов*

Описание формата	Выводимое число	Результат
# #	12	12
###	12	12
	12	12.00
	123	123.00
	.123	0.12
	12	12.
	1234	1,234.
	1.2	$1.20
$$##.##	1.2	$1.20
	123	*123.00
	1	***1.00
	1	* * $1.00
+ ####	123	+ 123
+ ####	-123	-123
	123	123
	-123	-123
##.##↑↑↑↑	123	1.23E+02
##.##↑↑↑↑	.012	1.20E-02

Обычно ведущие нули подавляются, а незначащие нули после десятичной точки печатаются. Исключение представляют собой числа, меньшие 1, при изображении которых печатается *0,* стоящий непосредственно перед десятичной точкой.

Для исключения возможности впечатать лишние цифры в чеки и векселя можно печатать вплотную к первой цифре числа знак доллара ($) или фунта стерлингов (£), для чего эти знаки указываются в формате удвоенными, и заполнять звездочками все оставшиеся пустыми места, указывая в форма-

те **. Комбинации *S или *£ обеспечивают сочетание плавающего положения знака денежной единицы и заполнения пустого места звездочками. Если положение знака денежной единицы является плавающим, то печатать знаки + и — во многих системах запрещается. Если требуется печатать отрицательные значения, то знак должен помещаться после числового поля (справа от него).

При использовании экспоненциальной формы представления чисел в научно-инженерных программах для печатания полной спецификации E+nn надо включать в формат все четыре знака карата (стрелка вверх). При этом первая из позиций, предназначенных для ведущих цифр, резервируется для изображения знака числа, так что "# #.# ↑↑↑↑" дает тот же эффект, что и "-#.#↑↑↑↑".

Запятая (если присутствует) может указываться в любом месте числового формата. Она используется для разделения изображения числа на группы по три цифры перед десятичной точкой.

Строковые форматы просты, но существует несколько систем описания строковых форматов, две из которых представлены в табл. 6.5. Система, обозначенная в этой таблице через "A", описывает выводимое поле как "<###", если надо напечатать до трех символов, выравнивая их по левому краю поля, и как ">###", если требуется выравнивание по правому краю поля. Примеры действия строковых форматов этой системы приведены в табл. 6.7.

В другой системе описания строковых форматов "B" строковое поле задается с помощью пары знаков обратной косой черты (\). Указанные слитно два этих знака задают печать двух символов строки. Указанные через один пробел — трех символов, и т. д. Знак & в строковом формате означает строковое поле с переменной длиной и обеспечивает вывод всей строки. Применение этого знака может нарушить регулярное расположение вывода по столбцам, являющегося основным достоинством оператора PRINT USING. Эту систему иллюстрирует табл. 6.8.

В некоторых системах для выравнивания влево до четырех символов строки поле задается как LLLL, а для выравнивания вправо до трех символов строки — как RRR.

Та блица 6.7. *Форматы строк символов в системе* "A"

Описание формата	Выводимая строка	Результат
	AB	AB
	ABC	ABC
	ABCD	ABC
	AB	AB
	ABC	ABC
	ABCD	ABC

Таблица 6.8. *Форматы строк символов в системе* "B"

Описание формата	Выводимая строка	Результат
!	ABCD	A
\\	ABCD	AB
\ \	ABCD	ABCD
\ \	AB	AB
&	ABCDE	ABCDE
&	A	A

Многие большие вычислительные системы руководствуются системой "A", в то время как системы с Бейсиком для микроЭВМ, например Microsoft, -системой "B".

Если число не помещается в числовом поле, то часто оно печатается целиком, но при этом в знак ошибки перед ним впечатывается символ %. Если строки не помещаются в поле, то лишние символы отбрасываются.

6.3.2. ПЕЧАТАНИЕ ПО ФОРМАТУ

Описание формата может содержаться в операторе PRINT USING, например:

100 PRINT USING "##.#"; A; B; C или в строковой переменной, например:
50 S$="##.#"
100 PRINT USING S$; A; B; C
что дает тот же самый эффект. Подобная гибкость позволяет при необходимости вводить наряду с данными и описания формата с тем, чтобы различные наборы данных можно было правильным образом изображать одной и той же программой.
В одной из систем для большой ЭВМ требуется, чтобы в том случае, когда формат находится вне оператора PRINT USING, он задавался в отдельной строке наподобие константы, без кавычек, но с предшествующим ему двоеточием. Номер строки с описанием формата должен указываться после USING следующим образом:
90 PRINT USING 200,A; B,C
200 :##.#
По-видимому, основной причиной выбора подобного способа послужило желание избежать совмещения длинного описания формата и списка переменных в одной строке с ограниченной длиной (операторы Бейсика переносить на другую строку нельзя) .
Оператор PRINT USING
Общая форма записи:
PRINT USING *описание формата, список выводимых данных*

где
описание формата - либо группа символов описания формата, заключенная в кавычки, либо строковая переменная, содержащая символы описания формата;
список выводимых данных - обычный список переменных и разделителей, используемых в операторе PRINT.
Учтите, что разделитель между описанием формата и списком выводимых данных от системы к системе меняется. Наиболее употребительны следующие формы записи:
описание формата, список выводимых данных описание формата; список выводимых данных описание формата: список выводимых данных
Детали употребления символов описания формата приведены в табл. 6.5. Значения элементов списка выводимых данных печатаются по порядку с использованием полей формата, указанных в его описании.
Если полей в описании формата больше, чем выводимых значений, то лишние поля игнорируются. Если полей в описании формата меньше, чем выводимых значений, то описание формата начинает использоваться заново от его начала до тех пор, пока не будут напечатаны все выводимые значения.
Приведенная ниже программа иллюстрирует некоторые из возможностей, предоставляемых оператором PRINT USING:

```
10 REM ДЕМОНСТРАЦИЯ ДЕЙСТВИЯ ОПЕРАТОРА PRINT USING
20 FOR I=1 TO 6
30   READ A$, A
40   PRINT USING "ЦЕНА \      \ =$ + ##.##" ; A$,A
50 NEXT I
60 DATA ПАЛЬТО,89.95.ПЛАТЬЯ.25.50
70 DATA РУЧКИ,1.99.ЧЕРНИЛ,.65
80 DATA ЧАСОВ,7.99,ОБУВИ,20
90 END
RUN
```

ЦЕНА ПАЛЬТО =$+69.95 ЦЕНА ПЛАТЬЯ =$+25.50 ЦЕНА РУЧКИ =$+1.99 ЦЕНА ЧЕРНИЛ =$+0.65
ЦЕНА ЧАСОВ =$+7.99 ЦЕНА ОБУВИ =$+20.00 END AT LINE 90

6.4. УПРАВЛЯЮЩИЕ СТРУКТУРЫ

В этом разделе рассматриваются приемы, позволяющие осуществлять четкое, эффективное и упорядоченное управление в программах на Бейсике. Обсуждаемые здесь формы управления довольно близки к тем, которые рекомендуются в структурном программировании.

Обсудив в предыдущих разделах проблемы, связанные с ограниченностью числа значащих цифр, рассмотрим теперь разновидность оператора, образую-

щего условное выражение в таких управляющих структурах, как IF, WHILE, UNTIL:

IF A = B THEN ...

Если значения входящих в условное выражение (A = B) переменных A и B получаются в результате длительных вычислений, то на равенство этих значений во всех ожидаемых случаях вряд ли можно рассчитывать. Поэтому условие может быть, а может и не быть выполненным в тех ситуациях, где программист рассчитывает иметь равенство. Чтобы поправить дело, либо пользуйтесь целыми величинами, например

IF A% = B% THEN

либо устанавливайте предел для разности значений, соответствующий числу значащих цифр для данного типа переменных. Если ожидается от шести до семи значащих цифр, то более надежной является проверка

IF ABS(A-B) <= 1.0E-6 THEN

6.4.1. НЕСКОЛЬКО ОПЕРАТОРОВ В ОДНОЙ СТРОКЕ

Во многих версиях Бейсика допускается указание нескольких операторов в одной (физической) строке при условии, что они разделяются специальным символом. Обычно разделителем служит двоеточие (:), но иногда в качестве разделителя используется символ обратной косой черты (\). Таким образом, после исполнения операторов

```
10  PRINT "ЗДЕСЬ": PRINT "И": PRINT "ЗДЕСЬ."
20   PRINT "СЛЕДУЮЩИЙ"
```

будет получен следующий результат: RUN

```
ЗДЕСЬ
И
ЗДЕСЬ.
СЛЕДУЮЩИЙ
```

Указанные после двоеточий операторы не имеют номеров, так что к ним нельзя перейти с помощью оператора GOTO. Они выполняются по порядку слева направо после выполнения первого оператора, помеченного номером строки.

Запись нескольких операторов в одной строке можно применять просто Для экономии числа знаков в набираемом тексте, но при таком подходе получаются программы, которые выглядят сплошной массой символов, не имеющей выраженной структуры. С другой стороны, эту возможность можно использовать для облегчения чтения программы и иллюстрации ее струк-

туры. Например, в необходимых местах можно добавлять оператор комментария:

```
200   FOR I=1 TO N1                  :REM ОБРАБОТКА ПО СТОЛБЦАМ
210     GOSUB 750                    :REM ВЫБОР ВЕДУЩЕГО ЭЛЕМЕНТА
220     I1=I+1
230   FOR J =I1 TO N                 :REM ИСКЛЮЧЕНИЕ В СТОЛБЦЕ I
```

Важнее то, что можно конструировать серии или блоки операторов, внутрь которых не может быть передано управление извне:

```
200   A=0: B=6:I=I+1: PRINT Z(I);
```

Тем самым образуется процесс со структограммой в виде прямоугольника:

```
A = 0
B =6
I=I+1
ПЕЧАТЬ Z(I) ;
```

Конечно, было бы лучше, если бы такой блок мог размещаться в нескольких строках, но, по крайней мере, небольшие блоки можно конструировать в одной строке.

Вышесказанное особенно полезно в операторах IF-THEN: 100 IF A> = B THEN PRINT "БОЛЬШЕ": Z=A: GOSUB 500

Если условие выполнено, то по порядку выполняются все операторы, следующие за THEN. Рассмотрим оператор

```
100   IF A>= B THEN PRINT "БОЛЬШЕ": GOSUB 500: Z=A
```

При его выполнении возникает следующая проблема: оператор Z=A никогда не будет исполнен, поскольку возврат после GOSUB произойдет к оператору в следующей строке, имеющему номер, а не к следующему оператору текущей строки. То же имеет место, если GOSUB заменить на GOTO. Небольшими циклами FOR-NEXT, помещающимися в одной строке, можно пользоваться для выполнения временных задержек:

```
100   REM ПАУЗА, ПРОДОЛЖИТЕЛЬНОСТЬ ЗАВИСИТ ОТ СИСТЕМЫ 110   FOR I=1 TO 10000: NEXT I
```

или для присваивания массивам начальных значений:

```
100   FOR I=1 TO 200:A(I)=0:NEXT I
```

или для выполнения многих других простых процессов.

6.4.2. СТРОКИ С ПРОДОЛЖЕНИЯМИ ОПЕРАТОРОВ

В немногих системах, например в Бейсике Microsoft, предоставляется средство записи логической строки в виде нескольких физических строк. Для этого в Бейсике Microsoft вместо завершения набора строки нажатием на клавишу возврата каретки надо набрать символ прогона строки (CONTROL J) и продолжить набор оператора на следующей строке:

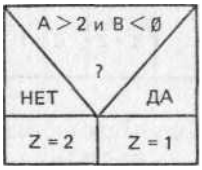
```
10 A=10*B+                    <CONTROL J>
Z*(C+2*B)                <ВОЗВРАТ КАРЕТКИ>
20  PRINT A                   <ВОЗВРАТ КАРЕТКИ>
```

Таким образом, одна логическая строка может занимать несколько физических строк, если при ее наборе пользоваться специальными управляющими символами. Обычно существует предел для общего числа символов в логической строке; типичное значение такого предела 255 символов. В Бейсике BBC набор строки можно продолжить, даже если достигнут ее конец: курсор автоматически переместится на начало следующей строки. Нажимать на клавишу возврата каретки надо только в конце логической строки.

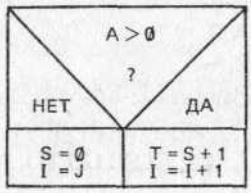
Подобная возможность выгодна при написании длинных операторов типа PRINT USING или IF. Кроме того, ее можно сочетать с написанием нескольких операторов в одной строке для образования блоков операторов с большим размером (см. подразд. 6.4.1).

6.4.3. ОПЕРАТОРЫ IF-THEN-ELSE

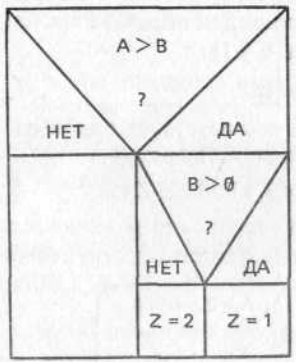
Оператор IF-THEN-ELSE соответствует одной из основных конструкций структурного программирования. Оператор
IF A>2 AND B<0 THEN Z = 1 ELSE Z=2
присвоит Z значение 1, если условное выражение истинно, и 2, если это выражение ложно. Он в точности соответствует структограмме



Используя разделители между операторами, выполняемые альтернативы можно сделать небольшими блоками. Так, оператору
IF A>0 THEN T=S+1: I=I+1 ELSE S=0:I=J
соответствует структограмма



Однако при этом весь оператор должен уместиться на одной строке, что представляет собой довольно серьезное ограничение. Но если текст оператора можно переносить на другие строки, то можно включать в каждый блок больше действий и добиваться удобочитаемого размещения текста. Например, в Бейсике Microsoft допустима следующая форма записи:
210 IF A>0 THEN REM ДОБАВИТЬ К СУММЕ <CONTROL J>
S=S+A: I=1+1 <CONTROL J>
ELSE REM ВЫЙТИ С НУЖНЫМ ЗНАЧЕНИЕМ <CONTROL J>
T=S*S <ВОЗВРАТ КАРЕТКИ>
В принципе операторы IF-THEN-ELSE могут быть и вложенными, но при их написании упоминавшиеся выше ограничения на длину строки оказываются еще более стеснительными. Понять серию вложенных операторов IF при записи в одну строку почти невозможно. Кроме того, такая запись требует особой аккуратности, так как, например, действие оператора
IF A>B THEN IF B>0 THEN Z = l ELSE Z=2 в Бейсике Microsoft представляется структограммой



поскольку ELSE связывается с ближайшим к нему THEN.
В Бейсике BBC служебное слово ELSE рассматривается как разделитель, расщепляющий строку на две части, отвечающие возможным результатам (ИСТИНА или ЛОЖЬ) вычисления условного выражения оператора IF. Тем самым в одной логической строке разрешается наличие только одного ELSE, хотя при этом текст оператора можно размещать на нескольких (физических) строках, например:

100 IF A>B THEN PRINT A,B
ELSE PRINT B,A <ВОЗВРАТ КАРЕТКИ>
Кроме того, ELSE можно добавлять к операторам ON-GOTO и ON-GOSUB для того, чтобы выполнять определенные действия по умолчанию, например:
ON I+J GOTO 100, 200, 300, 400: ELSE GO TO 1000

ON I+J GOSUB 100, 900, 200: ELSE PRINT "ОШИБКА": STOP
6.4.4. ИТЕРАЦИИ
Единственными формами операторов управления циклами, предусмотренными в стандарте Бейсика, являются цикл FOR-NEXT и комбинации IF-THEN и GOTO. Однако известны две основные формы управления циклами, а именно цикл REPEAT-UNTIL (повторять до), представляемый структограммой

где проверка условия завершения цикла осуществляется в его конце, и цикл WHILE (пока ..., выполнять), представляемый структограммой

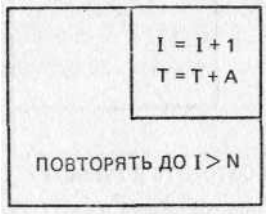
где проверка условия завершения цикла осуществляется в его начале. На псевдокоде эти циклы можно описать следующим образом:
ПОВТОРЯТЬ операторы ДО условие и
ПОКА условие ВЫПОЛНЯТЬ операторы
Принципиальное различие между ними состоит в том, что цикл ПОВТОРЯТЬ ДО всегда выполняется по крайней мере один раз, в то время как цикл ПОКА может не выполняться ни разу, если его условие не выполняется.
Цикл FOR-NEXT надо рассматривать как частный случай цикла ПОКА с условием, что значения счетчика цикла лежат в определенном диапазоне.

Если это условие не выполнено, то повторение цикла прекращается. Однако в некоторых системах, например в Бейсике BBC, проверка выполнения условия осуществляется в конце цикла FOR-NEXT, так что подобная его реализация является частным случаем цикла ПОВТОРЯТЬ ДО. В действитель-

ности единственной итерационной структурой общего вида в Бейсике BBC является ПОВТОРЯТЬ ДО; поэтому и оказывается, что в этой версии Бейсика проверка делается в конце циклов. Таким образом, эти циклы всегда выполняются по меньшей мере один раз независимо от содержащихся в них условий. Пример следующего цикла в Бейсике BBC

```
10  REPEAT
20    I=I+1
30    T=T+A
40  UNTIL I>N
```

представляется структограммой



а цикл

```
10  REPEAT
20  ......
30  ......
40  ......
50  UNTIL FALSE
```

оказывается бесконечным, так как его условие никогда не выполняется. (В Бейсике BBC есть две функции (FALSE и TRUE), возвращающие значения ЛОЖЬ и ИСТИНА соответственно.) Операторы цикла можно указывать в одной строке:

```
10  REPEAT INPUT A : SUM = SUM+A : UNTIL A<0
```

Версия Бейсика Microsoft служит примером другой итерационной структуры, ПОКА. В этой системе проверка условий как в цикле WHILE, так и в цикле FOR-NEXT осуществляется в начале цикла, а оператор REPEAT-UNTIL отсутствует. В версии Microsoft цикл начинается с оператора * WHILE условие затем следует группа операторов, а концом цикла служит оператор

```
WEND
```

В приведенном ниже примере использована программа пузырьковой сортировки (см. подразд. 4.5.1). Упомянутая программа могла производить не

являющиеся необходимыми проверки значений элементов уже упорядоченного массива, пока не выполнятся все N-1 проходов внешнего цикла.


```
1000 REM ФРАГМЕНТ ПРОГРАММЫ ПУЗЫРЬКОВОЙ СОРТИРОВКИ
1010 REM ПРЕДПОЛАГАЕТСЯ, ЧТО ЗНАЧЕНИЯ СОДЕРЖАТСЯ
1020 REM В МАССИВЕ A<1).....A(N) И СОРТИРУЮТСЯ
1030 REM В ПОРЯДКЕ УБЫВАНИЯ
1034 I=N
1038 SWAP=1         :REM ИСТИНА - НЕ РАВНО НУЛЮ
1040 WHILE SWAP
1045   SWAP=0
1050   FOR J=l TO I-1
1060     IF A(J)>=A(J+1) THEN 1100
1070     TEMP=A(J)
1080      A(J)=A(J+1)
1090     A(J+1)=TEMP
1095     SWAP=1
1100   NEXT J
1105   I=I-1
1110 WEND
1120 REM КОНЕЦ СОРТИРОВКИ
```

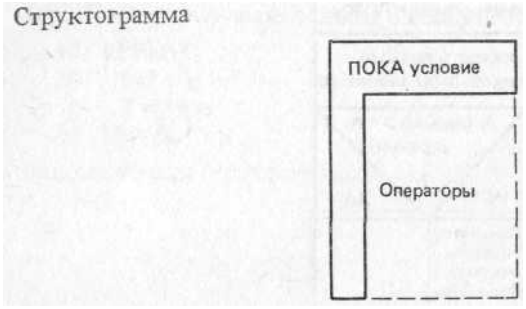
Если конструкция WHILE в системе отсутствует, то ее можно реализовать с помощью цикла REPEAT, поскольку действия операторов

WHILE условие WEND

полностью эквивалентны действиям операторов

IF условие THEN REPEAT

UNTIL NOT условие



соответствующая конструкции WHILE, эквивалентна


Это довольно запутанное преобразование, и поэтому в случае, если какая-либо из двух итерационных конструкций отсутствует, ее лучше всего имитировать подходящей комбинацией операторов IF-THEN и GOTO.

Некоторые авторы рекомендуют внутри цикла применение конструкции, которую на псевдокоде можно записать как "ЕСЛИ условие, ТО ВЫХОД". Ее действие состоит в преждевременном завершении цикла, если выполнено указанное в ней условие. По существу, она представляет собой не что иное, как передачу управления за пределы цикла первому оператору, стоящему за концом цикла. Таким образом, при достижении этого оператора не известно, произошло ли это в результате завершения цикла после выполнения его условия или же в результате выполнения условия в операторе IF. Хорошие методы разработки не допускают такой неопределенности, к тому же подобные действия могут вызывать трудности при теоретическом доказательстве правильности программы. Поэтому лучше всего не пользоваться псевдокодом подобного типа, как бы эта конструкция не казалась соблазнительной для применения на практике. Кроме того, она не имеет удовлетворительного

представления в виде структограммы. По этим причинам всегда предпочтительнее включать различные логические флаги или ключи в состав условия завершения цикла и изменять их значения внутри цикла, чтобы обеспечить его завершение.

6.5. ПРОЦЕДУРЫ И ВСТАВКИ

Не во многих системах предлагаются настоящие процедуры, представляющие собой совокупности команд, в известной мере похожие на блочные функции, однако в большинстве версий Бейсика для микроЭВМ предусматриваются определенные команды для вызова и исполнения из программы на Бейсике вставок на машинном коде.

6.5.1. ПРИМЕНЕНИЕ МАШИННОГО КОДА

В принципе применение в программах на Бейсике вставок, написанных на машинном коде, является не такой уж трудной задачей, так как во многих системах предусмотрен, по крайней мере, один способ вызова таких вставок. Однако на практике при этом приходится сталкиваться с множеством мелких деталей, например с адресом ячеек памяти и списками числовых значений, которые надо правильным образом устанавливать при каждом вызове вставки. Этого и следовало ожидать, так как машинный код имеет дело с физической реальностью ЭВМ; поэтому и приходится отыскивать и

использовать фактические адреса ячеек памяти, загружать значения в арифметические регистры, проверять и изменять указатели стека.
В большинстве систем предусмотрены две команды PEEK (взглянуть) и POKE (поместить), роль которых трудно переоценить. Эти команды позволяют облегчить выполнение указанных выше действий. Но ими надо пользоваться только в тех ситуациях, когда в Бейсике не предусмотрены средства присоединения вставок на машинном коде. Эти команды идут совершенно вразрез со всеми представлениями о разработке программ на языке высокого уровня, и пользоваться ими в общем случае не рекомендуется.
Команды PEEK и POKE
Общая форма записи: POKE I, J *(оператор) и*
переменная = PEEK (I) *(функция) где*
I *содержит десятичный (или шестнадцатеричный, если указан соответствующий символ) адрес байта памяти ЭВМ в диапазоне 0... 65 535;*
J *содержат десятичное (или шестнадцатеричное) значение, которое надо поместить в байт с адресом* I. *Значение* J *должно лежать в пределах 0. .. 255.*
Учтите, что в Бейсике BBC этих функций нет.

Таким образом, оператор 10 POKE 16251,22
поместит значение 22 в байт памяти с адресом 16251. Можно пользоваться и переменными; если они не являются целыми, то их значения будут округлены до ближайших целых чисел;

```
100   A=62000
110   B=33
120   POKE A,B
```

Некоторые системы позволяют пользоваться шестнадцатеричными значениями, перед которыми для отличия других величин указываются символы & или &H. Например, &FF представляет собой шестнадцатеричный эквивалент числа 255, а & 1A — шестнадцатеричный эквивалент 26. Таким образом, оператор

```
10   POKE & 5A00,&FF
```

поместит 255 в ячейку с адресом 23040.
Команда PEEK покажет Вам значение любого байта (8 бит) памяти. Ее применение иллюстрируется следующей программой:

```
10 REM ЧТЕНИЕ СОДЕРЖИМОГО ПАМЯТИ
20 INPUT "ВВЕДИТЕ НАЧАЛЬНЫЙ АДРЕС"; I%
30 INPUT "ВВЕДИТЕ ЧИСЛО БАЙТОВ";B%
40 FOR N%=0 TO B%-1
50   PRINT " " ;PEEK (I%+N%);
60 NEXT N%
70 PRINT
80 END
```

После ввода начального адреса и требуемого числа байтов программа распечатывает в десятичном виде (в диапазоне 0 ... 255) значения байтов памяти. Используя оператор PRINT " "; CHR$ (PEEK(1%+N%)); можно получить символьное представление значения каждого байта. В версии Бейсика Microsoft предусмотрена функция HEX$ (X), которая возвращает шестнадцатеричный эквивалент десятичного значения X, и ею можно воспользоваться вместо CHR$. Таким образом, заполненные значения можно очень легко изобразить в десятичном, символьном и шестнадцатеричном виде.

Обратной процедурой по отношению к чтению значений блоков байтов памяти служит запись новых значений в область памяти с помощью POKE. Она представляет собой удобный способ вставки небольшого числа машинных команд в программу на Бейсике. Например, программа

```
10   REM ДЕМОНСТРАЦИЯ ДЕЙСТВИЯ POKE
20   DATA  31,167,208,165,179.166
30   DATA 180,133,180,134,179,162
40   DATA   0,161.179,168,138, 76
50   DATA 120,210
60   FOR I%=0 TO 19 70   READ N% 80   POKE I%+826,N%
90   NEXT I%
100   END
200
```

занесет законченную вставку на машинном коде в область памяти с адресами от 826-го до 845-го для последующего вызова.

Во многих версиях Бейсика предусмотрено два способа вызова вставки на машинном коде. Наиболее общим способом является применение оператора CALL (вызов), имеющего следующий вид:

CALL адрес, входные параметры (входные параметры могут отсутствовать), например: 10 CALL&FF10

В версии Бейсика Microsoft для микроЭВМ с микропроцессором 6502 вызов вставки имеет несколько другую форму:

```
10   START=&FF10
20   CALL%START (10,20,30)
```

Исполнение команд передается Бейсиком вставке на машинном коде, начинающейся с заданного адреса. Если ей требуется передать какие-либо значения, то они могут быть загружены в некоторые из регистров, или в регистры могут быть занесены ссылки на эти значения, или эти значения могут быть переписаны в определенную область памяти, на которую будет иметься ссылка в паре регистров. Детали процесса передачи значений зависят от того, на каком именно микропроцессоре выполнена Ваша система. Если необходимо, Ваша вставка на машинном коде должна будет первым делом сохранить текущие значения регистров и указателей стека, а непосредственно перед возвращением в Бейсик восстановить эти значения с тем, чтобы Бейсик мог нормально продолжить свою работу с того места, откуда была вызвана вставка.

Другой способ вызова вставки на машинном коде состоит в том, что вставка используется как простая функция с единственным входным значением (аргументом) и единственным выходным значением (результатом). Обращение к ней осуществляется с помощью функции USR(X), например:

```
10   A=USR (2*B+3)
```

Иногда требуется, чтобы входное значение было целым. Тогда в случае вещественного X оно округляется до ближайшего целого числа. В некоторых системах и результат должен быть целым числом. Так как заранее не известно, где располагается вставка, то перед вызовом вставки ее начальный адрес должен быть занесен в определенное место (какое именно, зависит от конкретной системы) с помощью команды POKE, например:

```
100   POKE 1,58 110   POKE 2,4 120   A=USR(B)
```

Для занесения 16-битового адреса в память с помощью команды POKE приходится расщеплять его на два 8-битовых байта, помещая обычно младшие адреса в первый из байтов. В приведенном выше примере в байт с номером 1 заносилось значение 58, что равно 3*16+10 или ЗА в шестнадцатеричной

записи, а в байт с номером 2 — значение 4, что равно 4 в шестнадцатеричной записи. Шестнадцатеричные значения 3А и 4 в байтах 1 и 2 соответственно дают адрес
4*256 + 3*16+ 10=1082
После того как USR передает управление вставке, начинающейся с этого адреса, последняя извлекает значение входного параметра (переменной В в приведенном выше примере) из определенной области памяти, на которую указывает пара резисторов. В версии Бейсика Microsoft дополнительно предусмотрено простое средство для передачи адреса: если указан оператор

```
10   DEF USR0 = 2400
```

то при любом последующем обращении к USR0 (X) используется десятичный адрес, указанный в DEF USR0. (После USR можно указывать одну цифру, так что доступно до десяти различных вставок.)
В системе BBC также предусмотрена функция USR, но последняя берет адрес вставки из входного параметра, указанного при вызове USR, например:

```
20   A=USR(Z%) и
50   PRINT USR (& 1000)
```

При вызове USR значения специальных переменных A%, X%, Y% и C% копируются в микропроцессорные регистры A, X, Y и однобитовый флаг (регистр) переноса C для последующего использования во вставке. Оператор CALL в Бейсике BBC имеет форму, похожую на описанную выше, но дополнительно допускает возможность такой же передачи параметров через регистры и флаг переноса, как в случае USR. Детали процесса передачи параметров вставке см. в руководстве по Бейсику BBC.
Учтите, что в системе BBC не предусматриваются команды PEEK и POKE, но зато предлагается очень полезная возможность вставки команд на ассемблере прямо в обычные операторы Бейсика. Одна или несколько команд на ассемблере указываются после открывающей квадратной скобки ([), которую можно поставить в любом месте до или после оператора Бейсика. Вставка завершается закрывающей квадратной скобкой (]), например:

```
120  INPUT SUM
130   [.SHIFT ROR A
140      INT
150      CPY 8
160      BNE SHIFT ]
170  PRINT SUM
```

Предусмотренные в Бейсике BBC операции косвенной адресации (?, ! и $) являются более гибкими, чем команды PEEK и POKE, но выполняют аналогичные действия, например:
A=?B эквивалентно A=PEEK (B)
?A=B эквивалентно POKE A,B

Операция ? применяется к одному байту, операция ! - к одному слову, состоящему из четырех байтов, операция $ — к строке символов.

6.5.2. ПРОЦЕДУРЫ

Подпрограммы, для работы с которыми используются операторы GOSUB и RETURN, имеют дело исключительно с глобальными переменными, доступными в любом месте программы. Перед входом в подпрограмму определенные переменные должны содержать соответствующие входные значения.
Однострочные функции могут пользоваться как значениями, указанными в списке параметров функции, так и значениями глобальных переменных. Формальные переменные из списка параметров в действительности являются локальными по отношению к функции и отличаются от одноименных

глобальных переменных. Блочные функции расширяют возможности применения локальных переменных, допуская определение ряда локальных переменных, обладающих аналогичным свойством. Но однострочные функции возвращают одно и только одно значение в то место, откуда их вызывают. В принципе также могут "поступать" и блочные функции, хотя они могут иметь доступ и к глобальным переменным и изменять их, если требуется возвращать несколько значений.

Процедуры доступны в нескольких системах с Бейсиком для больших ЭВМ, но среди микроЭВМ только система BBC дает возможность пользоваться процедурами, что выглядит следующим образом: вызов процедуры осуществляется по имени, например

```
100   PROCPICTURE (9,27,6, BIG)
```

Процедуре можно передавать любое число входных параметров. Соответствующие им (по типу и номеру) формальные переменные являются локальными по отношению к процедуре, и, кроме того, можно определить дополнительные локальные переменные оператором LOCAL. Процедуре доступны и глобальные переменные, но пользоваться ими надо осторожно, так как желательно сохранить существенную независимость процедуры. Значения, указанные в списке параметров, копируются в локальные параметры для последующего использования. Процедура имеет следующий вид:

```
500   DEF PROCPICTURE (A,B,C$)
510   LOCAL I%,S
520   PRINT B,C$
530   FOR I%=A TO B
800   END PROC
```

Она начинается оператором DEF PROC *имя* и завершается оператором ENDPROC. Как и блочные функции, процедуры могут быть рекурсивными.

Процедуры служат идеальным средством реализации модульного програм-

мирования, проповедуемого в этой и других книгах. Их следует предпочитать подпрограммам. Как и функции в Бейсике BBC, процедуры надо помещать за концом основной программы.

Параметры процедуры можно использовать для передачи ей входных значений; все те значения, которые должны служить результатом вызова процедуры, необходимо передавать через глобальные переменные (т. е. переменные, используемые в основной части программы). Если результатом является единственное значение, то вместо процедуры следует использовать блочную функцию. Ниже приведен пример процедуры PROCNUM, вычисляющей два значения (сумму двух чисел и максимальное число) и возвращающей их через глобальные переменные C и D соответственно:

```
10 INPUT A,B
20 PROCNUM(A,B)
30 PRINT C,D
40 END
50 DEF PROCNUM(X.Y)
60   C=X+Y                    :REM C ИСПОЛЬЗУЕТСЯ ДЛЯ ВОЗВРАТА ЗНАЧЕНИЯ
70   IF X>Y THEN D=X ELSE D=Y   :REM И D ТОЖЕ
80 ENDPROC
```

Приведем пример другой процедуры, которая ждет до тех пор, пока не будет нажата клавиша с символом, заданным в A$:

```
250   DEF PROCWAIT (A$)
260   PRINT "НАЖМИТЕ КЛАВИШУ"; A$; "ДЛЯ ПРОДОЛЖЕНИЯ"
270   REPEAT UNTIL GETS =A$
280   ENDPROC
```

6.6. ПСЕВДОГРАФИКА

Псевдографика представляет собой возможность изображения обычных текстовых символов и символов особого вида в произвольном месте экрана ВТУ. Как минимум, программа на Бейсике должна иметь возможность изображать все символы из набора ASCII, а также управлять перемещениями курсора, но лучше всего, если кроме этого можно изображать некоторые символы особого вида или отдельные точки.

Подобные возможности псевдографики предоставляются многими системами посредством оператора PRINT. При этом оказывается возможным программировать функции управления курсором и изображать любой символ в любой заданной точке экрана. Персональная ЭВМ PET фирмы Commodore имеет очень хороший набор графических символов, так что при работе на ней описанный метод позволяет добиваться вполне удовлетворительных изображений.

Другие системы стремятся предоставить более богатые графические возможности и позволяют высвечивать или гасить в произвольном месте экрана небольшие области, или точки, или небольшие отрезки прямых линий, а также изображать их в разном цвете. В таких системах предусматриваются специальные операторы, например PLOT (чертить).

6.6.1 ПРИМЕНЕНИЕ ОПЕРАТОРА PRINT

Системы, в которых графические возможности обеспечиваются применением обычного оператора PRINT (например, персональная ЭВМ PET), позволяют с его помощью выдавать любые символы, в том числе и управляющие, в результате чего можно получить все те эффекты, которые достигаются нажатиями клавиш на клавиатуре. Таким образом, с помощью оператора PRINT можно добиться стирания экрана, перемещения курсора в начальную позицию, влево, вправо, вверх и вниз, вставки и удаления символов. Обычно для этих целей используется функция CHRS ( ), которая способна возвращать символы с кодом, соответствующим нажатию любой клавиши. Если Ваша система обеспечивает подобные возможности, то в результате исполнения программы

```
10   FOR I=0 TO 255 20      PRINT CHR$(T);
30   NEXT I
40   END
```

на экране должны произойти все виды странных действий, поскольку описанная выше программа выдает весь репертуар действий, вызываемых при нажатии клавиш. Каждая система имеет свои коды для многих управляющих действий, так что для иллюстраций этих действий выберем одну из систем, а именно PET. В табл. 6.9 собраны все виды управляющих действий для PET. Хотя описания этих действий относятся к системе PET, тем не менее они дают общее представление о работе подобных систем. Не забудьте, что в системах, где псевдографика ориентирована на оператор PRINT, номер текущей позиции курсора после изображения очередного символа увеличивается на 1 (или большее число, а зависимости от разделителя - запятой или точки с запятой). Для проведения обычным способом (слева направо) горизонтальной линии можно использовать операторы

```
10   FOR I=1 TO 10
20     PRINT"-";
30   NEXT I
```

Если попробовать провести вертикальную линию посредством операторов

```
10  FOR I=1 TO 10
20    PRINT "I";CHR$ (17);
30   NEXT I
```

то в результате получится изображение, более похожее на наклонную линию:

Таблица 6.9. *Функции управления экраном в персональной* ЭВМ PET *фирмы* Commodore

Действие	Код ASCII
Перемещение курсора вверх	145
Перемещение курсора вниз	17
Перемещение курсора влево	157
Перемещение курсора вправо	29
Гашение экрана	147
Перемещение курсора в левый верхний угол	19
Вставка символа	148
Удаление символа	20

Поэтому в такой ситуации всегда надо осуществлять сдвиг курсора влево; операторы

```
10  FOR I=1 TO 10
20    PRINT "1";CHR$(157);CHR$(17);
30  NEXT I
```

изображают требуемую вертикальную линию. Так как оператор PRINT выводит очередной символ в той строке и в той позиции, где в настоящее время находится курсор, то таким способом можно передвигаться по позициям символов. При этом задать новый адрес на экране нельзя и для перемещения от одной позиции к другой надо использовать цикл наподобие

```
100  FOR I=1 TO N: PRINT : NEXT I
```

который перемещает курсор на N строк вниз, а также функцию TAB для перемещения курсора вдоль строки. Если же требуется перемещение вверх,

то надо либо использовать CHR$(145), либо переместить курсор в начальную позицию (левый верхний угол экрана) и опуститься вниз на M строк:

```
200    PRINT CHR$ (19) ; :FOR I=1 TO M : PRINT : NEXT I
```

Следующая программа нарисует прямоугольник со сторонами K и L:

```
10 REM ВЫЧЕРЧИВАНИЕ ПРЯМОУГОЛЬНИКА НА ПЭВМ PET
20 INPUT "ВВЕДИТЕ СТОРОНЫ K И L";K,L
30 PRINT CHR$(147)                          :REM ГАШЕНИЕ ЭКРАНА
40 PRINT "L";
50 FOR I=1 TO K: PRINT "-";: NEXT I
```

```
60 A$=CHR$(157)+CHR$(17)                    :REM КУРСОР ВЛЕВО И ВНИЗ
70 PRINT "I";A$;
80 FOR I=1 TO L: PRINT "I";A$;: NEXT I
90 A$=CHR$(157)+CHR$(157)                   :REM КУРСОР ВЛЕВО И ВЛЕВО
100 PRINT "L";A$;
110 FOR I=1 TO K; PRINT "-";A$;: NEXT I
120 A$=CHR$(157)+CHR$(148)                   :REM КУРСОР ВЛЕВО И ВВЕРХ
130 PRINT "L";A$
140 FOR I=1 TO L: PRINT "I" ;A$; : NEXT I 150 END
```

Она выглядит довольно солидно и иллюстрирует негативные стороны слишком плотной упаковки операторов в каждой строке. Здесь это было сделано для того, чтобы каждая строка выполняла определенную логическую функцию. Строка 40 изображает левый верхний угол, строка 50 - верхнюю сторону прямоугольника, строка 70 - первый верхний угол, а строка 80 - первую боковую сторону.

Система PET обладает особым свойством, благодаря которому функции управления курсором видны в строках символов. Например, оператор

```
PRINT "-■I■I";
```

совершает те же действия, что и оператор PRINT "-";CHR$(157);CHR$(157);

поскольку символ■I служит представлением (только в пределах строки символов!) функции, соответствующей нажатию клавиши перемещения курсора влево, что, со своей стороны, эквивалентно CHR$(157). Несомненно, пользователи системы PET достаточно хорошо знакомы с этим свойством, так как такие странные символы неожиданно возникают при попытках исправить ошибку в строке символов.

Системы, в которых возможности оператора PRINT расширены, предусматривают либо вариант

PRINT AT строка, столбец; список выводимых данных,

где "строка" и "столбец" - числа или выражения, определяющие положение любой позиции символа на экране, либо вариант

PRINT @N, список выводимых данных,

где N - число или выражение, задающее положения на экране. В последнем случае выбирается система сквозной адресации позиций символов, начиная от левого верхнего угла экрана и заканчивая первым нижним углом. Следую-

щий пример составлен для системы ZX81 фирмы Sinclair, в которой используется форма PRINT AT (печать в ...), и представляет собой программу, изображающую тот же прямоугольник, что и выше:

```
10   REM ВЫЧЕРЧИВАНИЕ ПРЯМОУГОЛЬНИКА НА ПЭВМ SINCLAIR
20   INPUT "ВВЕДИТЕ СТОРОНЫ ПРЯМОУГОЛЬНИКА К И L"
30   INPUT К
40   INPUT L
50   CLS
60   PRINT AT 2.1в;CHR$(135)
70   PRINT AT 2,11+K;CHR$(4)
80   FOR I=1 TO К
90   PRINT AT 2,18+1;CHR$(131) 100   PRINT AT 3+L,10+I; CHR$(3)
110   NEXT I
120   PRINT AT 3+L,ll+K;CHR$(l)
130   PRINT AT 3+L,10;CHR$(2)
140   FOR I=1 TO L
150   PRINT AT 2+I,ll+K;CHR$(5) 160   PRINT AT 2+1.18;CHR$(133)
```

```
170   NEXT I
180   REM ПОМЕСТИТЬ МИГАЮЩИЙ ЗНАК В ЦЕНТРЕ
190   LET X=2+K/2
200   LET Y=10+L/2
210   PRINT AT X.Y;"OK"
220  GOSUB 260
230 PRINT AT X.Y; CHR$(180); CHR$(176)
240 GOSUB 260 250   GOTO 210
260 REM ПОДПРОГРАММА ДЛЯ ПАУЗЫ
270 FOR I=1 TO 10
280 NEXT I
290 RETURN
300 END
```

Как верхняя, так и нижняя сторона прямоугольника изображаются с помощью одного и того же цикла в строках 80 ... 110. Аналогично боковые стороны прямоугольника изображаются с помощью цикла в строках 140... 170; нижние углы — с помощью цикла с операторами в строках 120 и 130. В центре этого прямоугольника, левый угол которого находится в столбце 10 строки 2, изображается вспыхивающая метка. Этот эффект достигается с помощью инвертированных символов ОК (черных букв на белом фоне), равных CHR$(180) и CHR$(176) в системе Sinclair. Подпрограмма регулирует частоту вспыхивания. При аккуратном исполнении можно добиться известного оживления изображения.

В случае если требуется изображать или перемещать по экрану довольно сложные объекты, надо использовать какой-либо удобный способ обращения к ним. Система с обычным оператором PRINT (без AT или @) позволяет одной строкой описать законченный объект, например очертания шахматной фигуры. Для этого в строку надо помещать символы управления курсором с помощью функции CHR$ ( ). Таким способом создаются строки, определяющие законченные объекты. Для получения изображения объекта во время исполнения программы достаточно поместить курсор в требуемую позицию и выдать с помощью оператора PRINT заранее составленную строку. По-

добные строки можно сконструировать и для уничтожения изображения объекта, так что перемещение объекта реализуется с помощью нескольких операторов PRINT. Системы с операторами PRINT AT отличаются от систем, имеющих только оператор PRINT, и во многом приближаются к системам с оператором PLOT. Для определения объекта требуется ряд команд PRINT AT, поэтому последние лучше всего сгруппировать в виде функции или подпрограммы, имеющей несколько входных параметров, задающих положение объекта.

Такой подход в духе модульного программирования позволяет отвлечься от нагромождения деталей процесса изображения объектов, сосредоточив их в строках символов или в программных блоках. После этого ими можно очень удобно пользоваться как элементами языка высокого уровня, что выражается в простоте разработки программы.

Некоторые микропроцессорные системы отображают экран ВТУ в определенную область своей памяти таким образом, что каждый байт памяти отвечает одной позиции символа на экране ВТУ. При изменении любого из этих байтов памяти немедленно изменяется соответствующий символ на экране. Например, экран персональной ЭВМ PET состоит из 25 строк по 40 столбцов, всего 1000 позиций символов, и отображается в область памяти с адресами от 32768 до 33 767. Переход от номеров строк и столбцов к адресу ячейки памяти осуществляется с помощью соотношения

адрес памяти = 32 768+ (столбец— 1) +40* (строка— 1)

Приведенная ниже программа изобразит символ с кодом X в С-м столбце R-й строки:

```
10  INPUT C, R, X
20  POKE(32 768+C-1+40*(R-1)),X
```

Выдача изображения на экран с помощью POKE происходит гораздо быстрее, ' чем с помощью PRINT, но пользоваться этим надо осторожно. (См. комментарий к описанию команд PEEK и POKE в подразд. 6.5.1.)

6.6.2. ВЫЧЕРЧИВАНИЕ ИЗОБРАЖЕНИЙ

Средства вычерчивания изображений, предусмотренные в Бейсике, пока еще ограничены по сравнению с теми, которые предоставляются специализированными пакетами машинной графики, но имеют сходные характеристики. Вычерчивание изображений обеспечивается такими функциями, как PLOT, MOVE, DRAW, и подобными им. Эти функции манипулируют лучшими определениями, чем те, что доступны на символьном уровне, и позволяют высвечивать или гасить на экране элементы изображения символа или очень маленькие части экрана (точки). При этом может быть доступен и выбор цвета изображения.

В основу вычерчивания изображения положена концепция пера, движущегося над плоскостью и оставляющего на ней отметки. Таким образом, если текущая позиция имеет декартовы координаты (X0,Y0) , то в результате вызова DRAW X1,Y1 перо переместится в новую позицию (XI,Y1), оставив след

в виде отрезка прямой линии. Наиболее существенные "примитивы" процесса вычерчивания таковы:

(а) Поставить отметку (точку) в заданной позиции.

(б) Перейти к позиции X, Y, не делая никаких отметок.

(в) Перейти к позиции X, Y, оставляя след в виде прямой линии.

(г) Начертить какой-либо специальный символ в позиции X, Y. Последняя операция (г) не является существенно необходимой, но для полноты системы операции (а) - (в) необходимы. Правда, если потребуется, то можно обойтись операциями (а) и (б), повторив операцию (а) для получения изображения прямой линии. Однако не так просто, как кажется, составить наклонную линию из последовательности точек или других графических элементов. Для обеспечения более качественного приближения к прямой линии требуется довольно сложный алгоритм; вероятно, поэтому операция (в) и не реализована в некоторых системах.

Другой альтернативой является представление процесса создания изображения как возведение здания из кирпичей, что равносильно тем же действиям, которые были описаны для оператора PRINT в подразд. 6.6.1. Согласно этому подходу диаграммы рассматриваются построенными из ряда точек или символов, порядок изображения которых не является существенным. Здесь нет концепции вычерчивания непрерывной линии или перемещения "пера". В этом случае надо создавать подпрограммы или функции для определения графических объектов и использовать их так, как это было описано в упомянутом подразделе для оператора PRINT.

Система ZX81 фирмы Sinclair служит примером системы, предусматривающей простейшие средства для вычерчивания и позволяющей применение обоих описанных выше подходов. Каждая позиция символа на экране ВТУ рассматривается как квадрат, который делится на 4 квадрата (рис. 6.2),

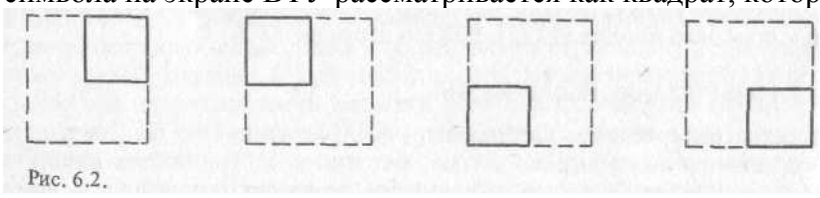


Рис. 6.2.

каждый из которых может быть высвечен отдельно с помощью вызова

PLOT X, Y и погашен с помощью вызова

UNPLOT X, Y

Так как всего имеется 22 строки по 32 столбца, то возможны значения Y (снизу вверх) от 0 до 43, а X (слева направо) от 0 до 63. Подпрограммы, реализующие перемещение к другой позиции и изображение отрезка прямой, могут быть получены из базовых операторов PLOT и UNPLOT. 210

Система BBC служит примером системы, предоставляющей полный набор настоящих средств вычерчивания. При работе с ней можно выбирать любой из пяти имеющихся режимов графики, отличающихся по определению и цветовой гамме. Все они основаны на декартовой системе координат, у которой начало находится в левом нижнем углу экрана, а размер координатной плоскости составляет 1280 единиц масштаба по горизонтали (по X-координате) и 1024 единицы по вертикали (по Y-координате) . В табл. 6.10 указано назначение различных операторов и функций.

Таблица 6.10. *Графические возможности в Бейсике* BBC

Графический оператор или функция	Действие
MODE N	Устанавливает один из восьми возможных режимов изображения данных на экране (от N = 0 до N = 7), но графика обеспечивается только в режимах N = 0, 1, 2, 4, 5
VDU список значений	Обеспечивает комплексные возможности управления графикой, цветом участков экрана, перемещением курсора. Например, VDU 22,N эквивалентно MODE N
GCOL N, M	Устанавливает основной цвет или цвет фона графического изображения и производит некоторые действия. При N = 0 значение M определяет цвет изображения: M =0-15 - основной цвет, M = 128-143 - цвет фона
CLG	Гасит графическое изображение на экране, закрашивает экран текущим цветом фона и перемещает курсор в исходную позицию с координатами (0, 0)
PLOT K, X, Y	Может выполнять различные действия в зависимости от значения K: K = 0 перемещение на (X, Y) по отношению к последней точке; K =4 перемещение в позицию с координатами (X, Y); K = 1,2,3 перемещение на (X,Y) по отношению к последней точке с изображением отрезка заданного цвета; K =5,6,7 перемещение в позицию с координатами (X,Y) с изображением отрезка заданного цвета; K =16-23 изображение пунктирного отрезка; K =64-71 изображение точки; K =80-87 изображение треугольника с вершинами в точке (X,Y) и в двух последних использованных точках
DRAW X, Y MOVE X, Y	Действует как PLOT 5, X, Y Действует как PLOT 4, X, Y

Система BBC предусматривает три типа операторов: выбор режима и других параметров с помощью оператора MODE или более сложного оператора
211

VDU, выбор основных цветов и инициация экрана с помощью операторов GCOL и CLG, а также полный набор средств вычерчивания, предоставляемый оператором PLOT. Операторы DRAW и

MOVE можно рассматривать как частные случаи оператора PLOT. Последующие расширения оператора PLOT смогут включать вычерчивание кривых.
В системе BBC команды COLOUR и CLS эквивалентны GCOL и CLG, но действуют только в текстовом режиме.
Треугольник для иллюстрации применения этих функций вычерчивает следующая простая программа:

```
10 REM ВЫЧЕРЧИВАНИЕ ТРЕУГОЛЬНИКА НА ПЭВМ BBC
20 REM ПРИСВАИВАНИЕ НАЧАЛЬНЫХ ЗНАЧЕНИЙ
30 ANGLE=2.094395        :REM 120 ГРАДУСОВ
40 CENTREX=640           :REM ЦЕНТР ТРЕУГОЛЬНИКА
50 CENTREY=512
60 R=400                 :REM РАДИУС ОПИСАННОЙ ОКРУЖНОСТИ
70 REM НАЧАЛЬНЫЕ УСТАНОВКИ ДЛЯ ВЫЧЕРЧИВАНИЯ
80 MODE 5
90 GCOL 0,129            :REM КРАСНЫЙ ФОН
100 GCOL 0,2             :REM ЖЕЛТЫЕ ЛИНИИ
110 CLG                  :REM ГАШЕНИЕ ЭКРАНА
120 REM
130 FOR THETA=0 TO 2 STEP .1 :REM ВРАЩЕНИЕ ТРЕУГОЛЬНИКА
140   X0=CENTREX+R*COS(THETA)
150   Y0=CENTREY+R*SIN(THETA)
160   MOVE X0.Y0
170 X=CENTREX+R*COS(THETA+ANGLE)
180 Y=CENTREY+B*SIN(THETA+ANGLE)
190   DRAW X,Y
200   X=CENTREX+R*COS(THETA+2*ANGLE)
210   Y=CENTREY+R*SIN(THETA+2*ANGLE)
220   DRAW X,Y
230   DRAW X0.Y0
240 NEXT THETA
250 END
```

6.7. ВО ЧТО ПРЕВРАЩАЕТСЯ БЕЙСИК

Не все версии Бейсика имеют расширения, описанные в предыдущих разделах настоящей главы. По-видимому, происходит так, что появляющуюся на рынке новую вычислительную систему стараются снабжать версией Бейсика, обладающей несколько бс5льшими возможностями, чем у конкурентов. Вследствие этого с годами язык разрастается, так как ранее сделанные расширения признаются большинством разработчиков компиляторов как стандартные свойства языка. Хотя многие из расширений имеют идентичное назначение, способы их записи значительно варьируются. В действительности существуют стандарт минимального подмножества Бейсика, разработанный Американским национальным институтом стандартов (ANSI) в 1976 году и принятый в 1978 году как стандарт ECMA-55 Европейской ассоциацией производителей ЭВМ, и последний стандарт ANSI, принятый в 1982 году и включающий в себя расширенные возможности.
Складывается впечатление, что расширения Бейсика возникают самым беспорядочным образом. Так, версия Бейсика Microsoft имеет только циклы

WHILE-WEND, а в Бейсике BBC циклы организованы по принципу REPEAT-UNTIL. Но если бы разработчики этих компиляторов поговорили с несколькими пользователями своих систем, то,

наверное, услышали бы пожелания иметь сразу оба типа структуры циклов. Это обусловило бы незначительное увеличение памяти, занимаемой компилятором/интерпретатором, и чуть-чуть замедлило бы его работу. Очевидно, фирма Microsoft решила реализовать цикл WHILE-WEND потому, что он похож на ее реализацию цикла FOR-NEXT, в которой проверка выполняется в начале цикла и, следовательно, компилятор/интерпретатор может обрабатывать эти циклы аналогичным образом. А в версии BBC проверка выполняется в конце циклов FOR-NEXT, чему хорошо соответствует конструкция REPEAT-UNTIL.

Одним из заслуживающих внимания этапов в развитии Бейсика является создание языка COMAL. Он был разработан в 1974 году в Дании Б. Кристенсеном и Б. Лофштедтом как расширение Бейсика. В настоящее время закончена работа над стандартом языка COMAL 80. Он предусматривает большое число расширений Бейсика и разработан с учетом современных идей построения языков программирования и методов структурного программирования и структурного проектирования. Этот язык прекрасно разработан, и некоторые из его свойств бегло описаны в следующем подразделе. Однако в последние годы было разработано немало хороших языков программирования, которые так и не получили широкого распространения не из-за каких-либо недостатков, а просто потому, что они не были поддержаны крупными производителями ЭВМ. Надо надеяться, что COMAL не постигнет эта судьба. Более подробные сведения об этом языке можно найти в книге Roy Atherton *Structured Programming with COMAL,* выпущенную издательством Ellis Horwood в 1982 году.

6.7.1. ОБЩИЕ СВОЙСТВА

Язык COMAL разрабатывался как серия расширений стандартного Бейсика. Он включает в себя стандартные управляющие структуры языков, подобных Алголу (например, Паскаля) , форма которых упрощена для большего соответствия принятому в Бейсике стилю записи операторов.

Операторы языка COMAL, выполняющие те же действия, что и большинство операторов Бейсика, должны располагаться в одной строке, но это правило не относится к управляющим структурам IF, CASE, FOR, WHILE, REPEAT и процедурным блокам, что снимает с программиста ряд нудных забот, возникающих при работе с Бейсиком.

Отличия языка COMAL от Бейсика достаточно отчетливы и усвоить их нетрудно: простые операторы типа присваивания должны целиком умещаться в одной строке, а расширенные операторы могут располагаться на нескольких строках. Тексту программы можно придавать форму, демонстрирующую ее структуру, чему немало способствует предоставляемая системой возможность отступов.

В языке COMAL присутствуют и выполняют в основном те же действия все знакомые по Бейсику операторы: INPUT, READ, DATA, RESTORE, PRINT, PRINT USING, IF-THEN, DIM, STOP, END. Но оператор GOTO требует,

чтобы метка имела имя, а не номер. Операторы программы имеют номера строк, обозначающие порядок их выполнения и помогающие редактировать программу, но наличие этих номеров не означает, что к этим операторам можно "перейти" так, как в Бейсике. Используя форму оператора GOTO, принятую в языке COMAL, можно "переходить" только к помеченным операторам, что уменьшает число возможных переходов и устраняет необходимость в операторе GOTO в наиболее типичных ситуациях.

В целом COMAL производит впечатление последовательного подхода, что нельзя сказать ни об одной версии Бейсика.

6.7.2. УПРАВЛЯЮЩИЕ СТРУКТУРЫ

Привычный цикл FOR-NEXT включен в COMAL в своей обычной форме, но с добавлением служебного слова DO (выполнять), указывающего на блочный характер цикла, например:

```
FOR LOOP=1 TO 10 STEP 2 DO
операторы
```

NEXT LOOP Этот оператор можно записывать в одной строке без указания NEXT:
FOR CNT = 1 TO 100 DO S=S + 10 B языке COMAL имеются циклы как WHILE, так и REPEAT, например:
WHILE A>0 DO
операторы
ENDWHILE и
REPEAT
операторы
UNTIL A>BOTTOM
а также однострочная версия цикла WHILE, похожая на FOR: WHILE (NOT A = B)DO A = A+C
Кроме того, имеется настоящий оператор выбора CASE, форма которого гораздо предпочтительнее, чем оператора Бейсика ON-GOTO; он позволяет осуществить выбор оператора по совпадению значений сложных выражений и допускает указания действий для непредусмотренных случаев с помощью служебного слова OTHERWISE (в противном случае) . Кроме того, для завершения оператора используется, опять-таки чтобы подчеркнуть его блочный характер, служебное слово ENDCASE (конец выбора) .

Предусмотрен и стандартный оператор IF-THEN, а также версия IF-THEN-ELSE, которая, как и все другие управляющие структуры, заканчивается соответствующим служебным словом (ENDIF) . Например, оператор
IF (A>B AND C=0) THEN
операторы ELSE
операторы
ENDIF имеет четкую и недвусмысленную форму.

6.7.3. ПРОЦЕДУРЫ

Процедуры имеют имена и допускают указание любого числа входных параметров. Процедура может действовать как подпрограмма или как функция; единственное отличие при этом состоит в способе вызова. Если процедура используется как функция, то внутри нее должно быть присваивание ее имени определенного значения. Вызов ее как подпрограммы осуществляется с помощью служебного слова EXEC, например:

```
20  EXEC BUFFEROUT(A,B)
```

а для вызова как функции имя процедуры указывается обычным образом в арифметическом выражении

```
100   DUMMY=LIMIT+MAX(A,B,C)
```

где MAX - имя процедуры, возвращающей значение. Структура такой процедуры может иметь следующий вид:

```
200   PROC MAX(REF X,REF Y, REF Z)
210     MAX=X
220     IF Y>MAX THEN MAX=Y
230     IF Z>MAX THEN MAX=Z
240   ENDPROC MAX
```

Обратите внимание на служебное слово REF при формальных параметрах X,Y,Z; оно указывает на то, что параметры A,B,C передаются по адресу. Это означает, что при манипуляциях внутри процедуры X,Y,Z действия осуществляются с фактическими параметрами A,B,C и что любые изменения X,Y,Z автоматически изменяют A,B,C. Следовательно, такие параметры при необходимости могут использоваться в качестве как входных, так и выходных. Массивы также передаются

по адресу; лишь простые строки или арифметические выражения передаются по значению, что осуществляется в виде односторонней передачи значения во временные переменные процедуры. В последнем случае нет никакой возможности возвратить значения основной програм-

ме. (Программисты, знакомые с Фортраном, могут протестовать, утверждая, что в этом языке возможен возврат значений из подпрограмм, при обращении к которым параметры передаются по значению. Да, это так, но такой возврат осуществляется очень своеобразным способом, при котором существует реальная опасность искажения констант и команд программы, что совершенно нельзя признать удовлетворительным.)

~ Таким образом, COMAL предусматривает очень полезные процедурные блоки и обеспечивает строгий подход к установлению соответствия между параметрами (фактическими и формальными) .

7. РАБОТА С МАТРИЦАМИ

В этой главе обсуждаются двумерные массивы, называемые *матрицами.* Матричная алгебра имеет дело с одномерными массивами, называемыми *векторами.*

Размеры всех векторов и матриц должны быть объявлены в операторе DIM до того, как ими будут пользоваться (см. подразд. 4.1.1) .Перед работой с массивами обратитесь в подразд. 4.1.2 относительно начального значения индексов, так как матричные функции обрабатывают только элементы массива с ненулевыми индексами.

Многие версии Бейсика для микроЭВМ не предусматривают описываемых в этой главе матричных функций, поэтому действия функций реализуются также посредством групп операторов Бейсика. Однако в последнем случае эти действия будут выполняться очень медленно, особенно при работе с большими матрицами (насчитывающими более 10*10 элементов). И если скорость выполнения оказывается критичной, то можно написать соответствующие этим действиям команды на машинном языке и для вызова написанной на машинном коде процедуры использовать оператор CALL, предусмотренный во многих версиях Бейсика. Для Вашей ЭВМ могут иметься в продаже готовые пакеты таких процедур.

7.1. ОПИСАНИЕ РАЗМЕРОВ

Массив, заданный определением DIM A(3), можно интерпретировать как вектор, например как вектор-строку

(a_1 a_2 a_3) или вектор-столбец

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Конкретный способ интерпретации зависит от того, как этот вектор используется в операторе умножения матриц. В системе эти два типа векторов не различаются.

Массив, заданный определением DIM B (3,4), можно рассматривать как матрицу 3*4:

$$\begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{pmatrix}$$

Первый из индексов в приведенном выше описании, 3, задает число строк, а второй индекс, 4, — число столбцов. Это соответствие выбрано так, что элементу b_{IJ} в алгебраической нотации отвечает B (I, J) при записи на Бейсике. Некоторые системы способны автоматически переопределять размеры. Хотя при этом общее число элементов в матрице не может быть изменено после того, как она была определена оператором DIM, но границы индексов менять можно. Например, матрица B (3,4) из приведенного выше примера имеет 12 элементов и может быть использована как B(2,5), имеющая 10 элементов, или как B (4,3), имеющая 12 элементов. Новые границы для индексов могут быть установлены в результате либо явного их определения, либо применения таких матричных операций, как умножение, сложение или обращение. Посмотрите в руководстве по Вашей системе, происходит ли в ней переопределение размеров.

7.2. МАТРИЧНЫЕ ЗНАЧЕНИЯ

7.2.1. ВВОД И ВЫВОД МАТРИЦ

В подразд. 4.1.6 достаточно полно обсуждалось применение операторов MAT INPUT и MAT PRINT, поскольку они очень полезны, даже если не пользоваться операциями над матрицами.

Оператор MAT INPUT

Общая форма записи:

MAT INPUT *матрица*

где "матрица" — имя (без указания скобок или индексов) одномерного или двумерного массива.
Система выдает приглашение к вводу данных, например вопросительный знак (?). *Данные можно набирать на клавиатуре, разделяя их запятыми. Если на одной строке набрано меньше данных, чем требуется, то будут выданы новые приглашения к вводу. Избыточные данные игнорируются. Обычное соглашение при вводе матриц состоит в том, что второй индекс изменяется быстрее, т. е. ввод происходит по строкам.*
В некоторых системах допускается указание в операторе ввода списка матриц и векторов, имена которых разделяются запятыми. По крайней мере, в одной системе в этом операторе вместо служебного слова INPUT *используется* READ.

Для имитации оператора MAT INPUT A можно использовать следующие операторы Бейсика:

```
10 DIM A(25)
20 FOR I=1 TO 25
30   INPUT A(I),
40 NEXT I
```

} MAT INPUT A

При работе с системой BBC уберите запятую из строки 30.
В некоторых системах предусмотрен оператор MAT LINPUT, осуществляющий ввод матриц и векторов, элементами которых являются строки символов. Этот оператор работает аналогично оператору LINPUT, описанному в подразд. 4.2.4.
Оператор MAT READ
Общая форма записи: MAT READ *матрица где "матрица " - имя одномерного или двумерного массива.*
Действует аналогично оператору MAT INPUT, *только берет значения из операторов* DATA.
Описание стандартных операторов READ и DATA см. в разд. 4.4. Оператор **MAT** PRINT *Общая форма записи:* MAT PRINT *список матриц*
где "список матриц" — одна или несколько матриц, имена которых разделены запятыми или точками с запятой.
Матрицы печатаются строка за строкой, все строки одной матрицы печатаются до того, как начнется печатание следующей матрицы. Если вектор указан внутри списка или если он указан в конце, но за ним следует разделитель, то вектор печатается в виде строки. Если вектор указан в конце списка и разделителя после него нет, то он печатается в виде столбца.
Если в качестве разделителя вместо запятой используется точка с запятой, то элементы каждой строки печатаются вплотную.
Управление форматом печати с помощью разделителей действует точно так же, как и в стандартном операторе PRINT. Приведем обычный эквивалент на Бейсике для оператора MAT PRINT A, B:

```
 10 DIM A(3,3),B(4)      ------
 20 FOR I=1 TO 3               |
 30   FOR J=1 TO 3             |
 40     PRINT A(I,J),          |
 50   NEXT J                   |
 60   PRINT                    > MAT PRINT A,B
 70 NEXT I                     |
 80 FOR I=1 TO 4               |
 90   PRINT B(I)               |
100 NEXT I                     |
110 PRINT                ------
```

7.2.2. ИНИЦИАЦИЯ МАТРИЦЫ

Предусмотрено несколько операторов для присваивания всем элементам матрицы одного и того же значения, 0 или 1, или для присваивания всем диагональным элементам квадратной матрицы значения 1. Последующим умножением на скалярное значение (см. подразд. 7.3.1) всем единичным элементам можно придать любое требуемое значение.

Операторы инициации матрицы

Общая форма записи:

MAT *матрица* = CON MAT *матрица* = ZER MAT *матрица* = IDN

где "матрица" - имя матрицы или вектора. Учтите, что в случае IDN *допускается только квадратная матрица.*

Указание CON *вызывает присваивание всем элементам матрицы или вектора единичного значения; указание* ZER *присваивает им нуль. Квадратные матрицы можно инициировать, указывая* IDN, *в результате чего всем диагональным элементам (от левого верхнего угла матрицы до правого нижнего) присваивается единичное значение, а остальным — нулевое.*

Некоторые системы позволяют явное переопределение размеров, указывая их в скобках после CON, ZER *и* IDN. *Например, в операторах*

MAT A = CON (6) MAT B = ZER (4,3)

матрица A *будет рассматриваться как вектор с размером* 6, *а матрица* B -*как имеющая размеры* 4*3.

Присваивания с указанием CON и ZER могут быть реализованы на Бейсике с помощью вложенных циклов FOR; например, MAT A = ZER можно имитировать следующими операторами:

```
10 DIM A(4,3)
20 FOR I = 1 TO 4
30   FOR J = 1 TO 3
40     A(I,J) = 0
50   NEXT J
60 NEXT I
```

MAT A = ZER

Присваивание с указанием IDN применяется к квадратным матрицам в форме MAT A = IDN. Оно может быть имитировано операторами

```
10 DIM A(6,6)
20 FOR I=1 TO 6
31   A(I,I)=1
40   FOR J=I+1 TO 6
50     A(I,J)=0
60     A(J,I)=0
70   NEXT J
80 NEXT I
```

MAT A=IDN

при условии, что в цикле FOR проверка делается в начале цикла. Если Ваша версия Бейсика этого не делает, то используйте пару вложенных циклов по аналогии с приведенным ранее примером для CON и ZER, добавив в него строку

45 IF I = J THEN A(I,I)=1 Приведем пример единичной матрицы с размерами 4*4:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

7.3. МАТРИЧНАЯ АЛГЕБРА

Операторы MAT обеспечивают выполнение всех основных операций над матрицами.

7.3.1. МАТРИЧНАЯ АРИФМЕТИКА

В отличие от обыкновенных арифметических операций, Бейсик не допускает сложных выражений с матричными операциями. Каждая строка должна содержать одну матрицу слева от знака

присваивания (=) и одну или две матрицы справа в зависимости от типа операции. В операторах MAT можно указывать как матрицы, так и векторы. В табл. 7.1 собраны существующие формы арифметических операций над матрицами.

Таблица 7.1. *Матричная алгебра. Переменные* A, B и C *могут быть векторами или матрицами;* к - *простая переменная, числовое выражение или константа*

Оператор	Действие	Комментарий
MAT A=B	Копирование элементов B в A	При переопределении размеры A заменяются на размеры B
MAT A=B+C	Поэлементное сложение B и C и копирование результатов в A	B и C должны иметь одинаковые размеры
MAT A=B-C	Поэлементное вычитание B и C и копирование результатов в A	B и C должны иметь одинаковые размеры
MAT A=(к)*B	Умножение каждого элемента B на скалярное значение к и копирование результатов в A	Скобки обычно обязательны
MAT A = B*C	Матричное умножение B и C в соответствии с правилами матричной алгебры	Число столбцов у B должно равняться числу строк у C

Оператор копирования, или замещения, MAT A = B присваивает значения элементов B(I,J) соответствующим элементам A(I,J). В некоторых системах размеры A переопределяются так, чтобы они совпадали с размерами B. Эквивалентная этому оператору программа имеет следующий вид:

```
10 DIM A(3,4),B(3,4)
20 FOR I=1 TO 3
30   FOR J=1 TO 4
40     A(I,J)=B(I,J)
50   NEXT J
60 NEXT I
```

MAT A=B

Операторы сложения и вычитания MAT A = B+C и MAT A= B-C требуют, чтобы B и C имели одинаковые размеры, и присваивают каждому элементу A(I,J) значения B (I, J) +C(I,J) и B (I,J)— C(I,J) соответственно. Программа
для сложения матриц имеет вид

```
10 DIM A(4,3),B(4,3),C(4,3)
20 FOR I=1 TO 4
30   FOR J=1 TO 3
40     A(I,J)=B(I,J)+C(I,J)
50   NEXT J
60 NEXT I
```

MAT A=B+C

Присваивание после выполнения этих арифметических функций может вызвать переопределение размеров A.

При скалярном умножении каждый элемент матрицы умножается на скалярное значение. Например, оператор
MAT A =(22.5)*B
вызывает присваивание каждому элементу A(I,J) значения 22.5*B(I,J). В некоторых системах требуется, чтобы скалярное значение было заключено в скобки; в других системах скобки не обязательны, но рекомендуются во избежание путаницы с умножением матриц. Приведем еще несколько примеров скалярного умножения:
MAT A=(X)*B
MAT A= (SQRT(T) +Y(Z)) *B
Оператору MAT A = (22.5) * B эквивалентна программа

```
10 DIM A(4,3) ,B(4,3)
20 FOR I = 1 TO 4
30   FOR J = 1 TO 3
40     A(I,J) = 22.5 * B(I,J)
50   NEXT J
60 NEXT I
```

MAT A = (22.5) * B

Матричное умножение является более сложным. Элементы MAT A=B*C задаются выражением
A(I,J) =B(I,1) *C(1,J) +B(I,2) *C(2,J)+ ... +B(I,M) *C(M,J)
$=^{M}\Sigma_{K=1}$ B(I,K)*C(K,J)
равным скалярному произведению I-й строки матрицы B на J-й столбец матрицы C.
.
Матрицы B и C не обязаны быть квадратными или иметь одинаковые размеры; требуется лишь совпадение числа столбцов у матрицы B с числом строк у матрицы C. Это условие вытекает из следующего соотношения для размеров:
размеры
A = B * C
(L*N) (L*M) (M*N)
B и C могут быть векторами или матрицами, а A присваивается полученный результат, например

$$\begin{pmatrix} 15 \\ 33 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} -1 \\ 5 \\ 2 \end{pmatrix}$$

Размеры 2*1 2*3 3*1

$$\begin{pmatrix} -4 & -2 & 0 \\ 33 & 45 & 57 \\ 13 & 23 & 33 \end{pmatrix} = \begin{pmatrix} 4 & -2 \\ 5 & 7 \\ 9 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Размеры 3*3 3*2 2*3
Ниже показаны примеры допустимых комбинаций (элементам матриц значения даны до умножения)
:

```
10 DIM A (2,2) , B (2,3), C (3,2) , D (3), E (2)
100 MATA = B*C
110 MAT E = B*D
120 MAT E = D*C
130 MAT D = E*B
140 MAT D = C*E
```

В строке 110 вектор D рассматривается как матрица с размерами 3*1, т. е. как вектор-столбец, и результат E имеет размеры 2*1 и тоже является векто-

ром-столбцом. Но в строке 120 вектор D рассматривается как матрица с размерами 1*3, т. е. как вектор-строка, и результат E имеет размеры 1*2, что отвечает вектору-строке. Как уже упоминалось, система не различает эти два типа векторов. Следующая программа служит примером умножения матриц:

```
 10 DIM A(3,5),B(3,8),C(8,5) ---
 20 FOR I=1 TO 3              |
 30   FOR J=1 TO 5            |
 40     S=0.0                 |
 50     FOR K=1 TO 8          |
 60       S=S+B(I,K)*C(K,J)   > MAT A=B*C
 70     NEXT K                |
 80     A(I,J)=S              |
 90   NEXT J                  |
100 NEXT I                   ---
```

Сложные операции, например 10 MAT A = (3)*B + C-D
в одной строке не допускаются и требуют преобразования в последовательность операторов
10 MAT A = (3)*B 20 MAT A = A + C 30 MAT A = A-D
Учтите, что деление матриц не определено; соответствующее делению действие может быть получено умножением на обратную матрицу.
Одна и та же матрица не должна появляться по обеим сторонам от знака (=) при умножении или при транспонировании, вероятно, из-за того, что в таком случае потребовалась бы дополнительная рабочая область в памяти. Так, операторы
MAT A = A*A
MAT A = A*B
MAT A = TRN(A)
недопустимы, в то время как возможны следующие операторы:
MAT B= A*A
MAT C=C+A
MAT D=B-D
7.3.2. ОПЕРАЦИИ НАД МАТРИЦАМИ
Другими широко распространенными операциями над матрицами являются обращение и транспонирование. Если система предусматривает автоматическое переопределение размеров, то она должна иметь определенные средства для получения информации о текущих рабочих пределах индексов массива. Например, в системе ICL 2903/4 предусмотрены две функции: ROW (A), возвращающая предельное значение первого индекса A, и COL(A), возвращающая предельное значение второго индекса A.

Транспонирование матрицы приводит к новой матрице, столбцами которой являются строки исходной. Если исходная матрица имела размеры N*M, то транспонированная матрица, обозначаемая через A, имеет размеры M*N. Для всех значений I и J выполняются соотношения $a_{IJ}=a'_{JI}$. Оператор Бейсика
MAT A = TRN(B) вызывает копирование элементов B(I,J) в A(J,I). Таким образом, матрица

$$A = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

является результатом транспонирования матрицы

$$B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Оператору MAT A = TRN(B) соответствует следующая программа на Бейсике:

```
1Ø DIM A(3,2), B(2,3)
2Ø FOR I = 1 TO 2
3Ø   FOR J = 1 TO 3
4Ø     A(J,I) = B(I,J)
5Ø   NEXT J
6Ø NEXT I
```

} MAT A = TRN(B)

Обращение может выполняться только над квадратными матрицами. Если A и B — такие квадратные матрицы, что

A*B=B*A=I

где I — единичная матрица, то B является обратной для матрицы A, что записывается в виде $B=A^{-1}$, а A является обратной для B, что записывается в виде $A=B^{-1}$, например:

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 3 \\ 1 & 2 & 4 \end{pmatrix} \begin{pmatrix} 6 & -2 & -3 \\ -1 & 1 & Ø \\ -1 & Ø & 1 \end{pmatrix} = \begin{pmatrix} 1 & Ø & Ø \\ Ø & 1 & Ø \\ Ø & Ø & 1 \end{pmatrix}$$

Каждая матрица в левой части этого равенства является обратной для другой. Не все квадратные матрицы допускают обращение; те из них, которые не имеют обратной, называются особенными. Оператор Бейсика

MAT A = INV(B) присваивает A матрицу, обратную для B (т. е. $A=B^{-1}$) .

Формально для вычисления обратной матрицы надо взять присоединенную матрицу и поделить ее на определитель исходной матрицы. Однако на практи-

ке этот способ неприемлем. Дело в том, что при работе с матрицами обязательно следует принимать во внимание ошибки вычислений, вызываемые в основном потерей значащих цифр из-за ограниченной точности представления чисел в Бейсике (от шести до семи значащих цифр), а для выполнения таких внешне простых операций, как обращение матрицы, приходится выполнять много тысяч простых арифметических операций, каждая из которых может внести свой вклад в ошибку. Поэтому самым лучшим способом получения матрицы X, обратной для A, оказывается решение системы линейных уравнений A*X=I

Для решения такой системы уравнений известно много эффективных методов, обладающих устойчивостью и минимизирующих влияние ошибок вычислений. Если Вам требуется вычислить обратную матрицу, а оператор обращения отсутствует, то примените один из таких методов. Простой метод обращения представлен в разд. 7.5.

7.4. ПРИМЕРЫ ОБРАБОТКИ МАТРИЦ

Приведенная ниже программа иллюстрирует некоторые из простых операторов MAT, в частности ввод, вывод и арифметические операции,

```
10 REM ПРОСТОЙ ПРИМЕР РАБОТЫ С МАТРИЦАМИ
20 DIM A(4,4),B(3,3),C(3,3)
30 REM
40 MAT INPUT B,C
50 PRINT "МАТРИЦА B"
60 MAT PRINT B
70 PRINT "МАТРИЦА C"
80 MAT PRINT C
```

```
90 REM
100 MAT A=B+C
110 PRINT "МАТРИЦА A=B+C"
120 MAT PRINT A
130 REM
140 MAT A=B-C
150 PRINT "МАТРИЦА A=B-C"
160 MAT PRINT A
170 END
RUN
?9.10.11.12.13 ?14.15 ?16.17.18 МАТРИЦА  B
1                    2                    3
4                    5                    6
7                    8                    9
МАТРИЦА   C
10                   11                   12
13                   14                   15
16                   17                   18
225


            МАТРИЦА A=B+C
            11                   13          15
            17                   19          21
            23                   25          27
            МАТРИЦА A=B-C
            -9                   -9          -9
            -9                   -9          -9
            -9                   -9          -9
            END AT LINE 170
```

Может появиться тенденция считать матрицы похожими на обычные переменные, особенно при выполнении над ними простых арифметических операций. Но в случае скалярных величин *a* и *b,* например, справедливо тождество $(a-b)^2=a^2-2ab+b^2$. Однако для матриц A и B в общем случае $(A-B)^2$ не равно $A^2 - 2AB+B^2$. В частном же случае B=I, где I - единичная матрица, соотношение $(A—I)^2 =A^2-2A+I$ справедливо, учитывая, что AI=IA=A и $I^2 =I$. Приведенная ниже программа иллюстрирует это равенство для специфических матриц A:

```
10 REM ИЛЛЮСТРАЦИЯ МАНИПУЛЯЦИЙ С МАТРИЦАМИ
20 REM ВЫЧИСЛЯЕТСЯ (A-I)**2 И СРАВНИВАЕТСЯ
30 REM С A**2-2*A+I
40 DIM A(20,20),B(20,20),C(20,20)
50 REM ИСПОЛЬЗУЕТСЯ ПЕРЕОПРЕДЕЛЕНИЕ РАЗМЕРОВ (ЕСЛИ ДОСТУПНО)
60 REM ЧТЕНИЕ ИДЕТ ПО ИСХОДНОМУ РАЗМЕРУ МАТРИЦЫ
70 READ N
80 REM ОПЕРАЦИЯ ZER ПЕРЕУСТАНАВЛИВАЕТ РАЗМЕРЫ
90 MAT A=ZER(N,N)
100 REM ЧТЕНИЕ ИЗ ОПЕРАТОРОВ DATA С РАЗМЕРАМИ N*N
110 MAT READ A
120 REM СФОРМИРОВАТЬ B В ЕДИНИЧНУЮ МАТРИЦУ С ПЕРЕОПРЕДЕЛЕНИЕМ
125 REM РАЗМЕРОВ
```

```
130 MAT B=IDN(N,N)
140 REM СФОРМИРОВАТЬ C=(A-I), ВЗЯВ РАЗМЕРЫ ОТ А И В
150 MAT C=A-B
160 REM ПОЛУЧИТЬ (A-I)**2
170 MAT B=C*C
180 PRINT "МАТРИЦА (A-I)**2"
190 PRINT "---------------"
200 MAT PRINT B;
210 PRINT
220 MAT B=IDN(N,N)
230 REM СФОРМИРОВАТЬ C=A*A (УНИЧТОЖИВ ПРЕЖНЕЕ СОДЕРЖИМОЕ)
240 MAT C=A*A
250 КЕМ СФОРМИРОВАТЬ C=A**2+I
260 MAT C=C+B
270 REM СФОРМИРОВАТЬ A=2*A
280 MAT A=(2)*A
290 REM СФОРМИРОВАТЬ C=A*«2-2«A+I
300 MAT C=C-A
310 PRINT "МАТРИЦА A**2-2*A+1"
320 PRINT "---------------"
330 MAT PRINT C;
340 PRINT
350 DATA 4
360 DATA 1,3,7,3,1.4,1.5,5,2,6,1.3,4,9,1
370 END
226
```

RUN

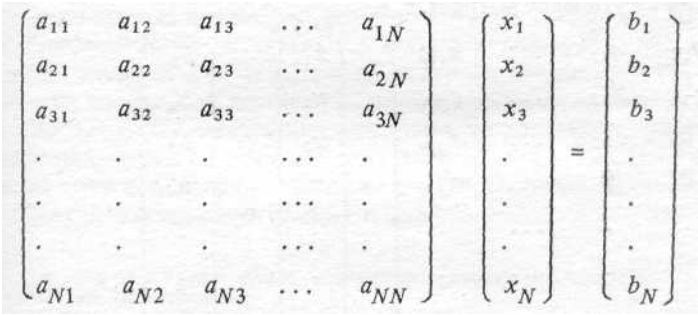
```
МАТРИЦА <A-I>**2
47 35 65 22
23 34 60 19
30 35 71 30
49 39 70 38
МАТРИЦА A**2-2*A+I
47  35  65  22
23  34  60  19
30  35  71  30
49  39  70  38
END AT LINE  370
```

Если переопределение размеров матриц Вашей системой не производится, то положите размеры массива равными (4,4), опустите строки 60, 70, 350 и измените строки 90 и 130, удалив из них список параметров (N,N).

Вы можете проверить, насколько общим является преобразование выражения $(A+B)^2$ в $A^2+2AB+B^2$, соответствующим образом модифицируя приведенную выше программу.

7.5. ЛИНЕЙНЫЕ УРАВНЕНИЯ

В этом разделе приводится законченная программа для решения системы линейных уравнений:
$Ax=b$, т. е.



по методу исключения Гаусса с выбором ведущего элемента по столбцам. Хотя этот метод прост, но выбор ведущего элемента по столбцам помогает сдерживать рост ошибок, вызванный потерей значащих цифр, что делает его очень полезным универсальным методом. Кроме того, его применение позволяет без труда вычислить обратную матрицу для A и определитель матрицы A.

Если Вы имеете дело с особой системой линейных уравнений (например, системой с большими размерами или плохо обусловленной, или с преобладающей диагональю, или с какими-либо иными особенностями), то обратитесь к литературе по численным методам, поскольку для решения систем линейных уравнений существует много различных приемов.

7.5.1. МЕТОД ИСКЛЮЧЕНИЯ ГАУССА

Приведем набросок этого метода, который будет воплощен в виде программы для ЭВМ:

(а) Выбрать наибольший элемент в первом столбце матрицы A.

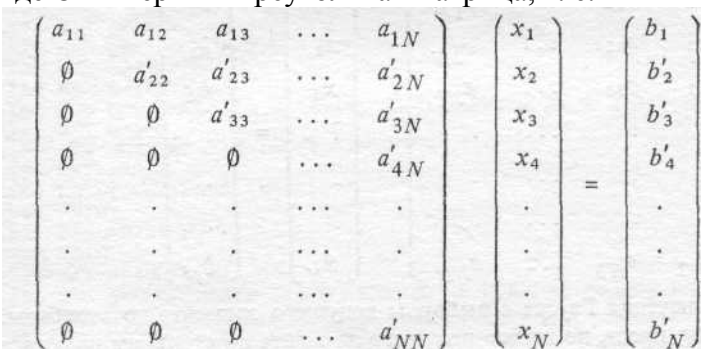
(б) Поменять местами первое уравнение (строку) с уравнением (строкой), содержащим выбранный элемент. При перестановке двух строк меняются местами и соответствующие элементы правой части *b;* так как порядок записи уравнений произволен, то при такой перестановке решение не изменяется. Эта операция называется выбором ведущего элемента со столбцами. Если наряду с перестановкой строк допускается и перестановка столбцов, то можно осуществлять выбор главного элемента, однако при таких перестановках трудно регистрировать порядок следования переменных.

(в) Вычесть из всех низлежащих уравнений такое кратное первого уравнения, чтобы в первом столбце всюду, кроме первой строки, образовались нули. При этом соответствующие множители будут равны (a_{21}/a_{11}), (a_{31}/a_{11}), (a_{41}/a_{11})и т. д. Проведенная в (б) перестановка минимизирует эти величины, что помогает уменьшить ошибки арифметических действий.

(г) Повторить все действия, начиная с (а), используя второе уравнение и второй столбец вместо первого. Продолжать эти действия для следующих уравнений и столбцов до тех пор, пока все элементы матрицы A, находящиеся ниже главной диагонали, не станут нулями. Тогда система уравнений примет вид

$Ux=b'$,

где U — верхняя треугольная матрица, т. е.



Такая система просто решается методом обратной подстановки. Последнее уравнение имеет вид

$a'_{NN}x_N= b'_N$,

откуда следует, что $x_N=b'_N\,a'_{NN}$- Метод обратной подстановки выполняется следующим образом:
(а) Вычислить из последнего уравнения x_N.
(б) Использовать полученное значение для решения предпоследнего уравнения, в котором отличны от нуля только коэффициенты При x_N и x_{N-1}.
228

(в) Использовать полученные значения x_N и x_{N-1} для решения предыдущего уравнения относительно x_{N_2}. Продолжать этот процесс для всех оставшихся уравнений, продвигаясь обратно к первому уравнению. При его достижении система уравнений решена и получены решения x_1 ,..., x_N.
Так как каждое уравнение (строка) не зависит от остальных, то его можно умножить или поделить на произвольное (ненулевое) число, не изменив решения системы. Поэтому прямое сравнение значений коэффициентов в столбцах матрицы А при поиске ведущего элемента не является наилучшим способом его выбора. Другой способ состоит в сопоставлении значений этих коэффициентов со значениями других коэффициентов в одном уравнении, сравнивая между собой вместо a_{ij} величины

$$\frac{a_{ij}}{\left(\sum_{k=1}^{N} a^2_{ik}\right)^{1/2}}.$$

Вычисления таких коэффициентов производятся подпрограммой в строке 990, а их сравнения и при необходимости перестановка уравнений - подпрограммой в строке 750. Попробуйте изменить эту подпрограмму так, чтобы выбор осуществлялся по непреобразованным значениям; попробуйте также удалить оператор GOSUB в строке 210. Тем самым выбор ведущего элемента будет устранен и можно будет оценить влияние выбора. Эффект особенно заметен для матриц, размеры которых превышают 10*10.
Дополнительные массивы C(,) и D( ) используются для сохранения исходных значений A(,) и B( ) в целях вычисления невязки (b-Ax). Отсутствие этих массивов не повлияет на решение системы, но благодаря их наличию можно получить очень важные сведения о точности решения. Информация о значениях потенциальных ведущих элементов и о перестановке уравнений также очень полезна для понимания поведения решаемой системы.
Ниже приводится программа GAUSS:

```
10 REM ПРОГРАММА GAUSS С ВЫБОРОМ ГЛАВНОГО ЭЛЕМЕНТА
15 REM ПО СТОЛБЦАМ
20 DIM A(6,6),B(6),X(6),C(6,6),D(6)
30 PRINT "РЕШЕНИЕ СИСТЕМЫ УРАВНЕНИЙ AX=B"
40 INPUT "ЧИСЛО УРАВНЕНИЙ";N
50 PRINT "ВВЕДИТЕ МАТРИЦУ А (СТРОКА ЗА СТРОКОЙ)"
60 FOR I=1 TO N
70   FOR J=l TO N
80     INPUT A(I,J),
90     C(I,J)=A(I.J) 100   NEXT J 110 NEXT I
120 PRINT "ВВЕДИТЕ ВЕКТОР B"
130 FOR I=1 TO N
140   INPUT B(I),
150   D(I)=B(I)
160 NEXT I
```

229

```
170 REM-----РЕШЕНИЕ СИСТЕМЫ-----
180 M=0
190 N1=N-1
200 FOR I=1 TO N1
210   GOSUB 750
220   I1=I+1
230   FOR J=I1 TO N
240     P=A(J,I)/A(I,I)
250          PRINT "КОЭФФИЦИЕНТ ДЛЯ СТРОКИ" ;J; "СТОЛБЦА" ; I "РАВЕН" ;P
260          FOR K=I1 TO N
270             A<J,K)=A<J,K)-P*ACI,K) ,
280          NEXT K
290          A(J,I)=0
300          B(J)=BJ)-P*B(I)
310       NEXT J
320  NEXT I
330 REM-----ОБРАТНАЯ ПОДСТАНОВКА-----
340 X(N)=B<N>/A(N,N)
350 I=N
360   I=I-1
370   I1=I+1
380   S=0
390   FOR J=I1 TO N
400     S=S+A<I,J)*X(J)
410   NEXT J
420   X(I)=(B(I)-S)/A(I,I)
430 IF I>1 THEN 360
440 REM ++++ПЕЧАТЬ РЕЗУЛЬТАТОВ++++
450 PRINT
460 PRINT "МАТРИЦА ПОСЛЕ ИСКЛЮЧЕНИЯ"
470 PRINT
480 FOR I=1 TO N
490   FOR J=l TO N
500     PRINT TAB(12*(J-l));A(I,J);
510   NEXT J
520   PRINT
530 NEXT I
540 PRINT
550 PRINT "ЧИСЛО ПЕРЕСТАНОВОК СТРОК =";M
560 PRINT
570 PRINT "РЕШЕНИЕ       НЕВЯЗКА"
580 PRINT "X                    <B-AX)"
590 FOR 1=1 TO N
600   S=0
610   FOR J=l TO N
620     S=S+C(I,J)*X(J)
630   NEXT J
640   PRINT X(I) ;TAB(16) ;D(I)-S
650 NEXT I
```

```
660 STOP
740 REM
750 REM ПОДПРОГРАММА ПЕРЕСТАНОВКИ СТРОК
760 REM ИЩЕТ НАИЛУЧШИЙ РАЗРЕШАЮЩИЙ ЭЛЕМЕНТ В СТОЛБЦЕ I
770 IF I=N THEN RETURN
780 L=0
790 FOR K=I TO N
800   GOSUB 990
810   PRINT "СТРОКА" ;K; "КОЭФФИЦИЕНТ" ;L1
820   IF L>=L1 THEN 850
830     I2=K
840     L=L1
850 NEXT K
230



860 IF I2=I THEN RETURN
870 M=M+1
880 REM ПЕРЕСТАНОВКА СТРОК I И I2
890 PRINT "ПЕРЕСТАНОВКА СТРОК" ; I 2 ; "И" ; I
900 FOR K=I TO N
910   S=A<I,K)
920   A(I,K)=A(I2.K)
930   A(I2,K)=S
940 NEXT K
950 S=B(I)
960 B(I)=B(I2)
970 B(I2)=S
980 RETURN
990 REM ПОДПРОГРАММА ОПРЕДЕЛЕНИЯ "ВЕСА" КОЭФФИЦИЕНТА В
995 REM СТРОКЕ К
1000 S=0
1010 FOR K1=I TO N
1020   S=S+A(K,K1)*A(K,K1)
1030 NEXT Kl
1040 L1=ABS(A(K,I)/SQR(S))
1050 RETURN
1060 END
RUN
РЕШЕНИЕ СИСТЕМЫ УРАВНЕНИЙ АХ=В
ЧИСЛО УРАВНЕНИЙ?3.
ВВЕДИТЕ МАТРИЦУ А (СТРОКА ЗА СТРОКОЙ)
?2.4.3
? 1.1.2
?3.5.2
ВВЕДИТЕ ВЕКТОР В
?0.2.1
СТРОКА 1 КОЭФФИЦИЕНТ .371391
СТРОКА 2 КОЭФФИЦИЕНТ .408248
```

```
СТРОКА 3 КОЭФФИЦИЕНТ .486664
ПЕРЕСТАНОВКА СТРОК 3 И 1
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 2 СТОЛБЦА 1 РАВЕН .333333
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 3 СТОЛБЦА 1 РАВЕН .666667
СТРОКА 2 КОЭФФИЦИЕНТ .447214
СТРОКА 3 КОЭФФИЦИЕНТ .371391
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 3 СТОЛБЦА 2 РАВЕН-1
МАТРИЦА ПОСЛЕ ИСКЛЮЧЕНИЯ
3                5                2
0               -.666667     1.33333
0                0                3
ЧИСЛО ПЕРЕСТАНОВОК СТРОК = 1
РЕШЕНИЕ       НЕВЯЗКА
 X                (B-AX)
3.16667           0
-1.83333          0
.333333           4.36557E-11
END AT  LINE  660
```

7.5.2. ОПРЕДЕЛИТЕЛЬ МАТРИЦЫ

Если в Вашей системе не предусмотрено вычисление определителя с помощью функции MAT, то одним из наиболее быстрых путей вычисления определителя матрицы A является ее преобразование к треугольному виду методом исключения Гаусса и перемножение всех элементов главной диагонали полученной матрицы.

Добавление к какой-либо строке матрицы кратного другой строки не изменяет значения ее определителя, а перестановка двух строк только меняет его знак. Поэтому в терминах элементов преобразованной матрицы

$$\text{Det (A)} = (-1)^S * a'_{11} * a'_{22} * a'_{33} * \ldots * a'_{NN}$$
$$= (-1)^S \prod_{i=1}^{N} a'_{ii}$$

где s — число перестановок строк в процессе исключения. Это значение печатается в строке 550, а произведение может быть получено исходя из изображения преобразованной матрицы, а еще лучше — путем добавления оператора в цикл FOR-NEXT, находящийся между строками 480 и 530. Ниже приводится результат исполнения программы по команде RUN:

```
OLD GAUSS
RUN
РЕШЕНИЕ СИСТЕМЫ УРАВНЕНИЙ AX=B
ЧИСЛО УРАВНЕНИЙ?4
ВВЕДИТЕ МАТРИЦУ A (СТРОКА ЗА СТРОКОЙ)
?1.4.-2.3
?2.2.0.4
?3.0.-1.2
?1.2.2.3
ВВЕДИТЕ ВЕКТОР B
?1.1.0.0          (В данном случае выбор произволен)
СТРОКА 1 КОЭФФИЦИЕНТ .182574
СТРОКА 2 КОЭФФИЦИЕНТ .408248
СТРОКА 3 КОЭФФИЦИЕНТ .801784
СТРОКА 4 КОЭФФИЦИЕНТ .235702
ПЕРЕСТАНОВКА СТРОК 3 И 1
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 2 СТОЛБЦА 1 РАВЕН .666667
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 3 СТОЛБЦА 1 РАВЕН .333333
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 4 СТОЛБЦА 1 РАВЕН .333333
СТРОКА 2 КОЭФФИЦИЕНТ .588348
СТРОКА 3 КОЭФФИЦИЕНТ .812743
СТРОКА 4 КОЭФФИЦИЕНТ .518321
ПЕРЕСТАНОВКА СТРОК 3 И 2
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 3 СТОЛБЦА 2 РАВЕН .5

КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 4 СТОЛБЦА 2 РАВЕН .5
СТРОКА 3 КОЭФФИЦИЕНТ .707107
СТРОКА 4 КОЭФФИЦИЕНТ .938343
ПЕРЕСТАНОВКА СТРОК 4 И 3
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 4 СТОЛБЦА 3 РАВЕН .473684
МАТРИЦА ПОСЛЕ ИСКЛЮЧЕНИЯ
 3            0           -1             2
 0            4           -1.66667       2.33333
 0            0            3.16667       1.16667
 0            0            0              .947368
ЧИСЛО ПЕРЕСТАНОВОК СТРОК = 3
РЕШЕНИЕ        НЕВЯЗКА
X              (B-AX)
-.666667        2.91038E-11
-.388889        2.91038E-11
-.444444        0
 .777778        2.91038E-11
END AT LINE 660
```

232

Таким образом, определитель
$(-1)^3$*3*4*3.16667* .947368= -36.000022
Точное значение равно -36, так что ошибка составляет 22 в седьмой и восьмой значащих цифрах.

7.5.3. ОБРАЩЕНИЕ МАТРИЦЫ

Если в Вашей системе обращение матрицы с помощью функции MAT не предусмотрено, то одним из практичных способов получения обратной матрицы является решение системы линейных уравнений со специальным набором векторов ее правой части.

Матрица B является обратной для матрицы A в том случае, если
A*B=I Теперь I можно рассмотреть как выбор таких векторов-столбцов e_i, что

$$\begin{pmatrix}1\\0\\.\\.\\.\\0\end{pmatrix} \quad e_2 = \begin{pmatrix}0\\1\\.\\.\\.\\0\end{pmatrix} \quad \ldots e_N = \begin{pmatrix}0\\0\\.\\.\\.\\1\end{pmatrix}$$

Таким образом, если Вы найдете такой набор векторов-столбцов b_1, b_2,. ...b_N, что
$Ab_1 = e_1$, $Ab_2 = e_2$,. .., $Ab_N = e_N$,, то обратной для матрицы A будет матрица $B=(b_1, \quad b_2 \ldots b_N)$

На практике существует два способа вычисления B. Если обращение матрицы требуется выполнять эпизодически, то используйте программу GAUSS в том виде, в каком она есть, и исполните ее N раз, по разу для каждого вектора e_i.

Если же Вы нуждаетесь в эффективной процедуре обращения, то модифицируйте программу так, чтобы вместо вектора B( ) использовалась матрица B(,). При этом исключение будет производиться

не N, а только один раз, но вместо отдельных значений надо манипулировать строками B(,). Обратная подстановка выполняется N раз, по разу для каждого столбца B(,).
233

Приведем пример, используя первый метод:

```
OLD  GAUSS
RUN
РЕШЕНИЕ СИСТЕМЫ УРАВНЕНИЙ AX=B
ЧИСЛО УРАВНЕНИЙ ?3.
ВВЕДИТЕ МАТРИЦУ A (СТРОКА ЗА СТРОКОЙ)
?3.0.1
ВВЕДИТЕ ВЕКТОР B
?1.0.0
СТРОКА 1 КОЭФФИЦИЕНТ .408248
СТРОКА 2 коэффициент .948663
СТРОКА 3 КОЭФФИЦИЕНТ .447214
ПЕРЕСТАНОВКА СТРОК 2 И 1
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 2 СТОЛБЦА 1 РАВЕН .333333
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 3 СТОЛБЦА 1 РАВЕН .333333
СТРОКА 2 КОЭФФИЦИЕНТ .514496
СТРОКА 3 КОЭФФИЦИЕНТ 0
КОЭФФИЦИЕНТ ДЛЯ СТРОКИ 3 СТОЛБЦА 2 РАВЕН в
МАТРИЦА ПОСЛЕ ИСКЛЮЧЕНИЯ
3                 0                 1
в                 -1                1.66667
а       в         1.66667
ЧИСЛО ПЕРЕСТАНОВОК СТРОК = 1 РЕШЕНИЕ       НЕВЯЗКА X                 (B-AX)
0                 0
-1                0
0                 0
END  AT LINE  660
```

Решение X представляет собой первый столбец обратной матрицы. Повторяя эту процедуру для векторов B, равных (0,1,0) и (0,0,1), получаем следующие (заключительные) результаты:

```
РЕШЕНИЕ                   НЕВЯЗКА
X                         (B-AX)
.4                        3.63798E-12
0                         0
-.2                       3.63798E-12
и
РЕШЕНИЕ                   НЕВЯЗКА
X                         (B-AX)
-2                        0
 1                        0
.6                        -1.45519E-11
```

соответственно. Таким образом, обратная матрица для

$$\begin{pmatrix} 1 & -1 & 2 \\ 3 & 0 & 1 \\ 1 & 0 & 2 \end{pmatrix}$$

имеет вид

$$\begin{pmatrix} 0 & 0.4 & -0.2 \\ -1 & 0 & 1 \\ 0 & -0.2 & 0.6 \end{pmatrix}$$

что можно проверить умножением этих матриц друг на друга, поскольку в результате должна получиться единичная матрица. Любые отклонения от нее указывают на степень точности обращения матрицы.

УПРАЖНЕНИЯ

7.1. Вычислите матрицу Z = A+B+C, где A, B и C - матрицы с размерами 4*4. Умножьте результат на матрицу D с размерами 4*3.

7.2. Эта задача предназначена для решения в системе, обеспечивающей переопределение размеров матриц с помощью операторов MAT. Пусть матрица A имеет размеры 10 * 10. Введите значение N, меньшее 10, и образуйте матрицу с размерами N*N, все элементы которой равны 5. Добавьте к ней матрицу с размерами N*N, у которой все элементы главной диагонали равны 5, а остальные равны 0, после чего вычислите и напечатайте обратную матрицу.

7.3. Пусть дана матрица

$$A = \begin{pmatrix} 1 & 2 & -1 & 1 \\ 2 & 0 & 1 & 2 \\ 1 & 0 & 1 & 3 \\ 2 & 3 & 1 & 1 \end{pmatrix}$$

Вычислите и напечатайте A^2, A^3 и A^4.

7.4. Пусть дана матрица

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 1 & -1 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

с размерами 3*3. Вычислите выражение $A^3 - 2A^2.- 9A$ и покажите, что оно равно нулю.

7.5. Решите с помощью программы GAUSS (из разд. 7.5) систему линейных уравнений

$19x_1+22x_2+42x_3 =25$

$27x_1 + 34x_2 +56x_3=18$

$52x_1+41x_2+17x_3=69,$

используя выбор ведущих элементов по столбцам и не используя его. Сравните два полученных решения.

7.6. Используя программу GAUSS (из разд. 7.5), вычислите определитель матрицы

$$A = \begin{pmatrix} 0 & 1 & -1 \\ 3 & 1 & -4 \\ 2 & 1 & 1 \end{pmatrix}$$

с размерами 3*3.

8. ФАЙЛЫ

Наличие архивной памяти того или иного рода существенно для любой вычислительной системы. Эта память представляет собой магнитный носитель, на который записывается содержимое памяти ЭВМ, включая как данные, так и программы, и который способен хранить информацию в течение длительного времени. Широко употребляются два типа носителей: ленты и диски, основные характеристики которых показаны в табл. 8.1. Ленты и лентопротяжные устройства обычно дешевле соответствующих дисков и поэтому могут обеспечить начальной архивной памятью вычислительную систему любой конфигурации.

Таблица *8.1.Некоторые характеристики носителей архивной памяти*

Носитель	Основное применение	Цена	Типичная емкость	Комментарий
Бытовая кассета с магнитной лентой	МикроЭВМ	Низкая	50К байт	Стандартная кассета для высококачественных бытовых магнитофонов
Спецкассета с магнитной лентой	Микро- и мини-ЭВМ	Средняя	10М байт	Применяется при профессиональной обработке данных и для регистрации данных
Бобина с магнитной лентой	Большие ЭВМ	Высокая	20М байт	Большие бобины с лентой шириной 1/2 дюйма (12,7 мм)
Гибкие минидиски	МикроЭВМ	Умеренная	300К байт	Диаметр *5%.* дюйма (133 мм), двусторонние, с двойной плотностью записи
8-дюймовые гибкие диски	Микро- и мини-ЭВМ	Средняя	1 Мбайт	IBM-стандарт, двусторонние, с двойной плотностью записи
Винчестерский несъемный диск	Микро- и мини-ЭВМ	Средняя	5М байтов	Диаметр 8 дюймов (203 мм), более жесткие, чем гибкие диски
Съемный жесткий диск	Большие ЭВМ	Высокая	100М байт	Емкость до 1000М байт

В первую очередь это относится к современным микроЭВМ, развитие которых позволяет провести тесную историческую параллель с развитием больших ЭВМ в течение последних двух десятилетий. Первоначально на больших ЭВМ в качестве архивной памяти применялись ленты, а очень ограниченные по емкости и дорогие дисковые устройства резервировались для постоянного хранения важных системных программ и временного хранения программ, исполнение которых приостановлено. По мере того как дисковые устройства становились более дешевыми и более емкими, ленты постепенно стали выходить из употребления. В настоящее время ленты начинают рассматриваться как архивная память для долговременного хранения копий дисков и как средство обмена данными между раз-

ными ЭВМ. А информация, требующая непосредственного доступа, хранится на дисках.

Параллельное развитие операционных систем обеспечило программистам возможность одинаковой работы с файлами независимо от типа устройства, с которым они связаны. Это означает, что все

типы файлов можно хранить на современных скоростных дисках с высокой плотностью записи. Например, некоторые файлы, известные как "ленточные", могут находиться на магнитном диске, но выглядеть как магнитные ленты для использующей их программы. Операционные системы микроЭВМ пока еще обладают минимальными возможностями, так как они должны полностью помещаться в памяти микроЭВМ и при этом оставлять как можно больше места для программ. В настоящее время разрабатываются более совершенные операционные системы, которые вскоре обеспечат микроЭВМ большинством возможностей, присущих большим ЭВМ.

Файлом именуется собрание элементов данных. Такое собрание, или файл, идентифицируется по имени, присвоенному его создателем. Операционная система способна ассоциировать имя файла с физической областью хранения его данных таким образом, что все детали оказываются скрытыми от программиста. Файлы могут быть организованы для последовательного или произвольного (прямого) доступа. В *последовательном* файле элементы данных хранятся в виде длинного непрерывного потока. За один прием считывается по одному элементу, и доступ к ним осуществляется последовательно. В файле с *прямым доступом* элементы данных хранятся независимо и доступ к любому из них осуществляется примерно за одно и то же время.

В разд. 8.1 обсуждаются программные файлы, а в разд. 8.2 даются представление о файлах данных и более детальное описание файлов различных типов, о которых уже говорилось. Специфике организации файлов на отдельных носителях и доступа к ним средствами Бейсика посвящены разд. 8.4 и последующие разделы.

8.1. ПРОГРАММНЫЕ ФАЙЛЫ

Написанные на Бейсике программы обычно запоминаются в ЭВМ в некотором внутреннем формате, который довольно близок к тому, какой изображается на экране ВТУ, но отличается тем, что в нем удалены пробелы, служебные слова заменены на условные символы, а переменные могут заменяться ссылками на таблицу условных символов. Этот компактный формат обычно используется при сохранении программ в файлах. Такие файлы организованы последовательным образом, причем программе предшествуют ее имя и другие детали. Файл может храниться на ленте или диске, и метод его использования достаточно стандартен.

В табл. 8.2 собраны команды, копирующие программы из памяти в файл и обратно. В целом команды SAVE и LOAD довольно стандартны, чего нельзя сказать об их некоторых дополнительных параметрах. В персональной вычислительной машине PET в этих командах надо указывать номер устройства ввода-вывода; если он отсутствует, то подразумеваются файлы на

Таблица 8.2. *Команды для работы с программными файлами*

Действие	Примеры команд	Комментарий
Сохранение копии программы в файле	SAVE "FRED" SAVE	Обычный для микроЭВМ формат. В команду могут входить дополнительные параметры В ICL 2904 такая команда сохраняет программу, которой предварительно присвоено имя (командами NAME FRED или NEW FRED)

Загрузка копии программы из файла	LOAD "FRED" GET FRED OLD FRED	Обычный для микроЭВМ формат. В персональной ЭВМ PET эта команда содержит дополнительный параметр, определяющий устройство (диск или магнитофон), а в Бейсике Microsoft в ней можно указать R для немедленного запуска программы Формат, принятый в ICL 2904 и многих других больших ЭВМ
Удаление программного файла	KILL "FRED" KILL FRED UNSAVE FRED	Используется только в дисковых системах для микроЭВМ Используется на ICL 2904 и многих других больших ЭВМ
Считывание другой программы с диска на место текущей программы	10 CHAIN "FRED" 10 LOAD "FRED"	Является оператором и может иметь много дополнительных параметров В некоторых системах, например PET, применение LOAD в качестве оператора дает тот же эффект, что и команда CHAIN

бытовой магнитной кассете. При работе с диском требуется указывать специальное имя файла, включающее в себя номер дисковода. Например, команда

LOAD "0:FRED",8

загрузит в память файл FRED с диска (устройство 8), установленного в дисковод 0.

При работе с бытовой магнитной кассетой такой команды удаления файла, как KILL (уничтожить), нет: подразумевается, что для удаления файла надо его перезаписать. Обычно протяжку ленты в кассете, запись и перемотку приходится инициировать вручную. Типичная последовательность действий при выполнении команды SAVE такова: надо набрать полный текст команды, но клавишу возврата каретки (или ENTER) пока не нажимать. Затем надо включить магнитофон и перевести его в режим записи. Спустя

несколько секунд нажимается клавиша возврата каретки (или ENTER), чтобы послать команду ЭВМ. Через несколько минут система выдаст на ВТУ сообщение ОК (все в порядке), после чего ленту можно остановить.

Кассетная система BBC запоминает детальные сведения о программе в каждом блоке программы, записанном на ленте. Эти сведения изображаются при загрузке программы в память с помощью команды LOAD и обеспечивают полезную индикацию состояния процесса загрузки. Кроме того, с помощью команды *CAT довольно просто организуется каталог содержания ленты при ее чтении от начала до конца. Эта возможность очень полезна при перемотке ленты до нужной программы. Достаточно набрать *CAT, пропустить несколько файлов или перемотать ленту ближе к началу и затем читать ленту до тех пор, пока не будет обнаружена требуемая программа.

Иногда из-за ограниченности памяти программа не может в ней поместиться целиком. Эта проблема устраняется оператором CHAIN (связывать в цепочку) , вызывающим другую программу для перезаписи текущей и передающим ей управление. Рассмотрим пример программы, написанной (конечно же!) в модульном стиле для выполнения вычислений и выдачи результатов в графическом виде. В памяти ЭВМ помещается только по одному модулю за раз, так что модули надо попеременно вызывать командой CHAIN (рис. 8.1) .

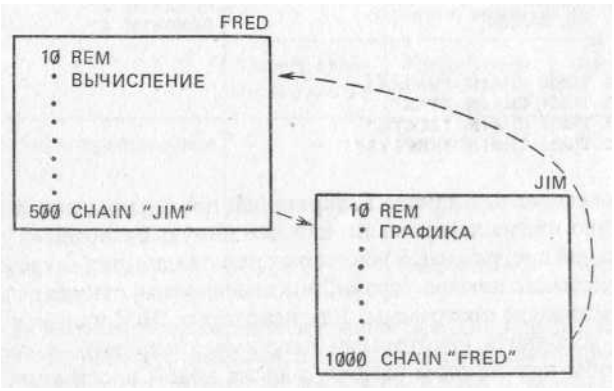


Рис. 8.1. Вызовы программ с помощью команды CHAIN

Исполнение программы начинается с модуля FRED. При этом запрашиваются и вводятся некоторые данные, над которыми выполняются определенные вычисления. Так как результаты требуется начертить на экране, то оператор 500 вызывает файл JIM для перезаписи текущей программы и продолжает исполнение с начала считанной программы построения изображений. После завершения программа JIM уничтожает себя, вызывая программу FRED, которая может запросить новые данные и продолжить работу или же сразу остановиться. Процесс смены программ протекает медленно, но часто иного пути исполнения больших программ нет.

Точные детали исполнения команды CHAIN меняются от системы к системе. Большинство систем требует, чтобы каждый сегмент включал в себя основную программу. Поэтому, к сожалению, производить смену только подпрограмм нельзя. Довольно часто исходные переменные сохраняют свои значения после смены программы, а в других случаях, например в Бейсике Microsoft, для указания тех переменных, значения которых требуется сохранить, используется оператор COMMON. Аналогичным образом многие системы позволяют сохранять и привязки к файлам данных, так что следующий сегмент может продолжать чтение или запись с того места, на котором остановилась предыдущая программа. В любом случае те значения, которые надо передавать от сегмента к сегменту, можно поместить во временные файлы данных, которые можно уничтожить в конце исполнения.

Широко распространенной общей программной конструкцией является меню, с которым удобно сочетаются операторы CHAIN:

```
10 REM ПРОГРАММА ДЛЯ РЕАЛИЗАЦИИ ОСНОВНОГО МЕНЮ
20 REM
30 PRINT "МЕНЮ ОПЕРАЦИЙ"
40 PRINT "------------"
50 PRINT "ВЫБЕРИТЕ, ПОЖАЛУЙСТА, ТРЕБУЕМЫЙ ПРЕДМЕТ" 60 PRINT "
ЧТОБЫ ЗАКОНЧИТЬ, ВВЕДИТЕ И"
70 PRINT "ЗАРПЛАТА                    ВВЕДИТЕ 1"
80 PRINT "НАЛОГ                       ВВЕДИТЕ 2"
90 PRINT "СЧЕТА                       ВВЕДИТЕ 3"
100 PRINT "СЛУЖАЩИЕ                       ВВЕДИТЕ 4"
110 INPUT N
120 REM
130 IF N=l THEN CHAIN "WAGES"
140 IF N=2 THEN CHAIN "TAX"
150 IF N=3 THEN CHAIN "ACCTS"
160 IF N=4 THEN CHAIN "EMPLYS"
```

170 END

В этой программе нет явного возврата для повтора выдачи изображения меню, но неявно предполагается, что каждый модуль возвращает управление в начало основной программы. В некоторых разновидностях оператора CHAIN допускается указание номера строки, показывающего, откуда надо начинать исполнение вызванной программы. Для некоторых ЭВМ приведенный выше пример надо дополнить некоторыми деталями; например, в системе PET требуется, чтобы при каждом вызове с диска новой программы командой CHAIN в ячейки 42 и 43 заносилось значение указателя на начало области переменных командой POKE.

Файлы любого типа содержат определенные средства обнаружения ошибок, связанных с потерей или искажением данных. Это особенно важно при работе с более чувствительными к повреждениям носителями — кассетами с магнитной лентой и гибкими магнитными дисками. Обычно к каждому запоминаемому блоку байтов добавляется несколько лишних битов, позволяющих системе обнаруживать ошибки при чтении блока. К несчастью, ошибка только в одном блоке может воспрепятствовать считыванию всего файла, а то и всего диска — по крайней мере, на имеющихся в Вашем распоряжении устройствах. Попробуйте в этом случае прочитать кассету или диск на другой

ЭВМ, так как ее устройства могут иметь другую чувствительность; если и это не помогает, а информация очень важна, то у поставщиков или производителей ЭВМ могут иметься средства, позволяющие спасти значительную часть
данных.

8.2. ФАЙЛЫ ДАННЫХ

В разд. 8.1 было показано, как можно хранить программы в архивной памяти ЭВМ. То же можно делать и с данными, но в этом случае программист должен уметь разбираться в различных типах структуры файлов и отражать эти особенности в программе. Обычно характеристики файла тесно связаны со свойствами физического носителя, поэтому эти свойства надо учитывать при описании разнообразных файлов, доступных с помощью Бейсика. В табл. 8.3 показано, с какими отдельными типами файлов можно встретиться в разд. 8.4 — 8.6 и в каких подразделах описаны вариации этих типов, обусловленные различными носителями информации.

Таблица 8.3. *Сочетания типов файлов и носителей их информации с указанием разделов настоящей главы, в которых они описаны. (На больших ЭВМ последовательными файлами обычно служат файлы прямого доступа, используемые последовательным образом)*

Тип файла	Носитель информации		
	Кассета с магнитной лентой	Гибкий диск	Диск большой ЭВМ
Последовательный в терминальном формате		Подразд. 8.5.2	Подразд. 8.5.1
Последовательный во внутреннем формате (разд. 8.4)	Подразд. 8.4.2	Подразд. 8.4.5	Подразд. 8.6.1
Прямого доступа (разд. 8.6)		Подразд. 8.6.2	Подразд. 8.6.1

Прежде чем поближе познакомиться с характеристиками файлов, рассмотрим общий метод работы с файлами, проиллюстрированный на рис. 8.2 и 8.3. Данные поступают в программу по имени PROCESS в результате ввода с помощью операторов INPUT. После выполнения некоторых вычислений программа записывает информацию в файл INFO, находящийся в архивной памяти. Другая программа, RESULTS, читает информацию из файла INFO и выдает результаты на ВТУ или на принтер (см. рис. 8.3). Распечатывание стандартными средствами файлов большинства типов не дает содержательных сведений ввиду того, что данные хранятся в них в особом внутреннем формате. Поэтому в качестве посредника между пользователем и файлом требуется своя программа. Из этого правила есть одно исключение: файлы одного типа, называемые файлами в терминальном формате, сохраняют информацию в форме строк символов в том виде, в каком они набираются на клавиатуре. Файлы этого типа можно стандартными средствами записывать с клави-

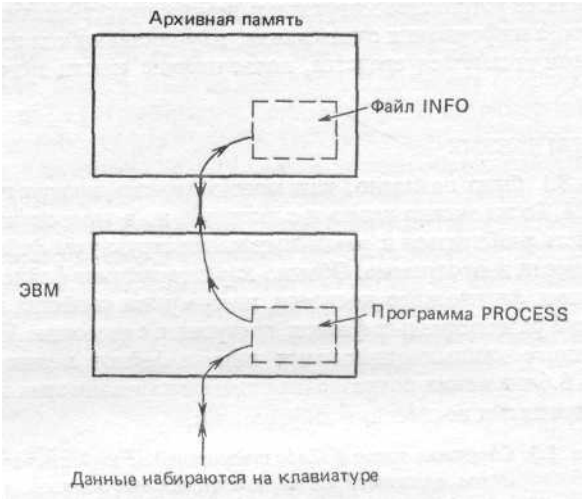


Рис. 8.2. Набираемые на клавиатуре данные воспринимаются программой PROCESS, которая запоминает их в файле INFO

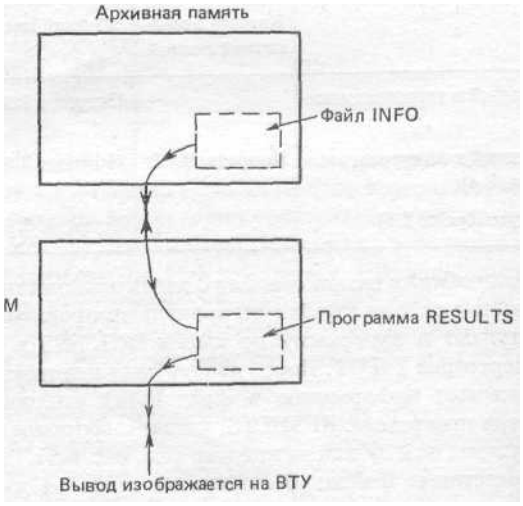


Рис. 8.3. Данные считаются из файла INFO программой RESULTS и выводятся на ВТУ

атуры непосредственно в архивную память и распечатывать прямо на ВТУ, однако с ними можно работать и программно описанным выше способом.

Зрительно подобный файл можно представить себе так, как показано на
8 4. Его можно считать похожим на изображение, выдаваемое на экран ВТУ. Любое чтение или перезапись файла должны происходить строка за строкой от его начала до конца. Таким образом, подобный файл организован последовательно, и способ хранения его данных лучше всего представлять

Рис. 8.4. Зрительное представление файла в терминальном формате

себе следующим образом: строки элементов данных примыкают друг к другу, как бусинки на нитке. На рис. 8.5 показана организация файла, изображенного на рис. 8.4, в терминальном формате. После каждой строки текста, запомненного в том виде, в каком он изображен на ВТУ, следуют символ возврата каретки (CR) и символ перехода на следующую строку (LF), которые при изображении обеспечивают переход на начало следующей строки. В разных системах приняты несколько различающиеся признаки конца строки, но многие системы используют для этого пару символов CR, LF. Другие типы последовательных файлов имеют такую же форму организации, но при этом изображенные на рис. 8.5 строки заменяются на блоки данных. При работе с последовательным

файлом нельзя извлечь строку (блок) из его середины, изменить ее и поместить обратно в тот же файл.
Но почему же нельзя? По идее, файл представляет собой непрерывный ряд значений и измененный элемент данных может оказаться короче или длиннее исходного. Однако на практике файл запоминается в виде ряда блоков на ленте или диске и оказывается невозможным прочитать нужную позицию в

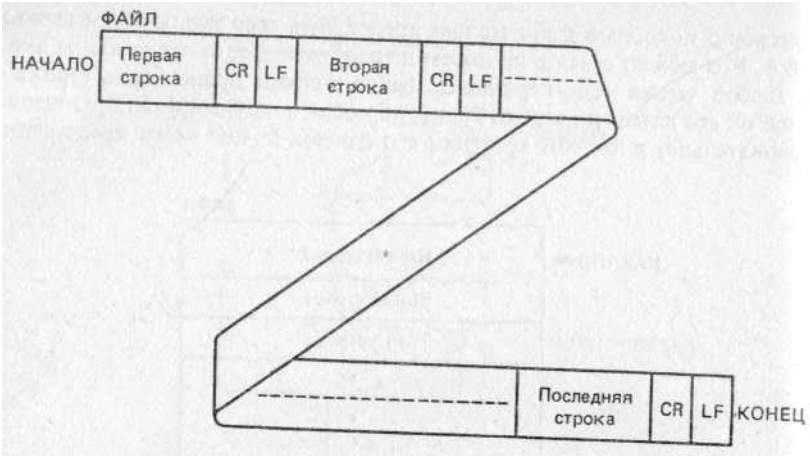


Рис. 8.5. Модель организации хранения данных файла в терминальном формате

файле, быстро переключиться в режим записи и снова вернуться к режиму чтения, не повредив при этом данные до и после записанной информации. Выполнение изменений в последовательном файле включает в себя копирование всех элементов исходного файла в новый файл, с попутным изменением требуемых значений. Последовательными файлами (как в терминальном, так и в других форматах) пользоваться легко, но кроме описанного выше неудобства модификации данных им присущи и другие недостатки, исключающие их применение в определенных ситуациях. Например, если подобный файл содержит две тысячи записей с фамилиями и адресами, отсортированных по фамилиям в алфавитном порядке, и от программы требуется найти адрес г-на Уилсона, который оказался в блоке данных с номером 1950, то для получения нужного блока надо пропустить все предшествующие 1949 блоков (т. е. прочитать их и проигнорировать). Этот процесс отнимает немало времени, что демонстрирует непригодность последовательной организации для применения в подобной справочно-информационной программе, хотя такая организация может оказаться идеальной для многих других приложений (например, для печати расчетных и инвентаризационных ведомостей, наклеек с адресами).

Тип файлов, организованных для прямого доступа к записям, позволяет преодолеть обе упомянутые выше трудности. На рис. 8.6 показано зрительное представление организации такого файла. Записи с данными хранятся на диске в соответствии с имеющимся у ЭВМ представлением о наличии свободного места. При каждом запросе программы на чтение или на запись должен быть указан номер записи, по которому во внутреннем справочнике находится ее фактическое положение на диске, что позволяет немедленно произвести чтение или запись. Таким образом осуществляется прямой доступ к записям, требующий примерно одного и того же небольшого (зависящего от скорости

Рис. 8.6. Модель организации хранения данных файла прямого доступа

вращения диска) времени для всех записей. Если файл с фамилиями и адресами из приведенного выше примера организован для прямого доступа, то программа сможет очень быстро найти адрес г-на Уилсона и, если потребуется, изменить его. У файлов прямого доступа есть свои недостатки. Обычно требуется, чтобы все записи файла имели одинаковую длину, а иногда еще и строго определенный формат. Внутренний справочник файла может лимитировать число доступных записей и, следовательно, размер файла. Работа с файлом прямого доступа может потребовать от программиста гораздо больших усилий. Дополнительная информация о работе средствами Бейсика с файлами последовательного и прямого доступа приводится в последующих разделах.

8.3. ХАРАКТЕРИСТИКИ НОСИТЕЛЕЙ ИНФОРМАЦИИ

Простейшим и самым дешевым носителем информации является бытовая магнитофонная кассета. Для работы с ней требуются высококачественные бытовые магнитофоны, да и сами кассеты должны быть высокого качества. Записываемая на магнитную ленту информация выводится из ЭВМ в виде группы символов (байтов) в кодах ASCII, передаваемых последовательно (друг за другом). Нули и единицы, содержащиеся в каждом байте, обычно представлены двумя различными звуковыми частотами; например, в системе BBC 0 представляется частотой 1200 Гц, а 1 - частотой 2400 Гц. Типичная скорость передачи данных составляет 300 бод, что соответствует передаче данных примерно 300 бит/с; поэтому для записи на ленту или считывания в ЭВМ программы размером 8К потребуется не менее 3,6 мин. Фактическое время несколько превышает расчетное за счет записи помимо программных данных еще заголовка и концевика, и на практике среднее время составляет 4—5 мин. Таким образом, кассета С60 (рассчитанная на 60 мин) способна хранить около двенадцати программ размером 8К каждая, или около 100К байт данных. Кассетные магнитофоны некоторых вычислительных систем обеспечивают более высокие скорости передачи данных вплоть до 1200 бод. При такой высокой скорости передачи данных они записываются на ленту, движущуюся с обычной скоростью, более плотно. Время чтения-записи при скорости передачи 1200 бод составляет четверть того, которое требуется при скорости передачи 300 бод, и лента с той же длиной способна вместить вчетверо больше информации.

Некоторые виды носителей информации должны при работе находиться в контакте с головками чтения-записи, что вызывает постепенный физический износ магнитного покрытия. К числу таких носителей относятся магнитная лента и гибкие диски. При работе с ними возможна потеря информации из-за износа их поверхности; поэтому целесообразно хранить по две или три копии каждого важного файла.

Гибкие диски бывают двух размеров: 8-дюймовый (203 мм) изначальный вариант, разработанный фирмой IBM, и популярный вариант с размером $5^1/_4$ дюйма (133 мм). Методы записи информации на 8-дюймовые диски устоялись и стали довольно стандартными; хотя диски размером *5%* дюйма физически похожи на 8-дюймовые, но различных методов записи для них гораздо больше. Как показано на рис. 8.7, гибкий диск похож на граммофонную пластинку со скоростью вращения 45 об/мин, вложенную в картонный конверт. Он представляет собой пластиковый диск (заштрихован на рис. 8.7), покрытый магнитным материалом и вращающийся в своем конверте со скоростью 360 об/мин. Для снижения трения конверт имеет подкладку из]

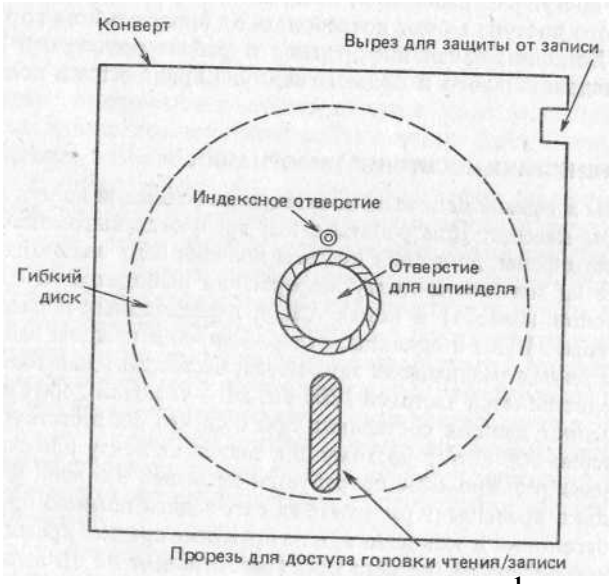


Рис. 8.7. Гибкий диск диаметром $5^1/_4$ дюйма (113 мм). Он вложен в конверт, из которого видна только небольшая часть поверхности диска (заштрихована)

покрытого графитом пластика. Данные записываются на концентрических круговых дорожках через специальную щель, вырезанную в конверте. Чтобы обеспечить точку отсчета начала каждой дорожки, на диске пробито отверстие, которое ощущается в тот момент, когда оно проходит мимо индексного отверстия конверта. Каждая дорожка представляет собой отдельную окруж-

ность (в отличие от спирального желобка граммофонной пластинки) и содержит примерно одно и то же количество информации, вследствие чего на дорожках, находящихся ближе к центру диска, информация записана плотнее, чем на дорожках, расположенных ближе к его внешнему краю. Одна дорожка на 8-дюймовом диске содержит около 5000 байт данных; чтение или запись одной дорожки занимает 1/360 мин, так что скорость передачи данных составляет около 240 000 бит/с. Не вся поверхность дорожки содержит "полезную" информацию; способы расположения данных на дорожке меняются от системы к системе, но по 8-дюймовому формату IBM каждая дорожка разбивается на 26 секторов по 128 байт данных в каждом. Тем самым на данные отводится $3^1/_4$ Кбайт, а остальную часть дорожки занимают заголовки, разделители данных, поля с контрольными суммами и заполнители.

Фактическая скорость передачи данных от ЭВМ к диску ниже расчетной, равной 19,5К байт/с, и действия аппаратуры и программного обеспечения ЭВМ, отвечающих за передачу данных, могут уменьшить ее до 0,5К байт/с.

Восьмидюймовый гибкий диск в формате ЮМ имеет 77 дорожек, так что полезная емкость одной стороны диска составляет около 250К байт, но часто вместо этой цифры называют 400К байт, учитывая и служебную информацию. Базовый вариант гибкого диска размером $5^1/_4$ дюйма имеет 35 дорожек полезной емкостью 2,25К байт каждая, что дает общую емкость одной поверхности 78К байт. Распространены все варианты двойной и учетверенной плотности, обеспечивающие большую емкость по сравнению с этим базовым вариантом. Один из способов повышения доли полезной информации на дорожке состоит в пробивке отверстий по внешнему краю диска, указывающих начало секторов с данными. Тем самым исключается необходимость записи этой информации на дорожки, в результате чего полезная емкость может возрасти на 25 %; этот метод известен как жесткая разметка диска (hard sectoring), а описанный ранее метод называется программной разметкой (soft sectoring).

Винчестерский жесткий диск появился недавно и ведет свое происхождение от больших жестких дисков, предназначенных для больших ЭВМ. Он состоит из одного или более жестких дисков диаметром 5 ... 14 дюймов (127 ... 356 мм), вращающихся в герметизированном корпусе. Вместе с дисководом и "электронной начинкой" небольшие модели винчестерских дисков по размерам не превышают дисководы для гибких дисков. Дисковый носитель не является съемным и, подобно дискам для больших ЭВМ, имеет летящую головку чтения-записи, плавающую над поверхностью диска на воздушной подушке толщиной около 20 микродюймов. Воздушная подушка создается за счет вращения диска со скоростью 2400 об/мин. Благодаря плаванию головки поверхность диска не изнашивается, а герметичный корпус исключает царапание головкой диска за счет попадания на него частиц пыли. Поэтому винчестерские диски в настоящее время являются очень надежными носителями информации.

Некоторые свойства последовательных файлов унаследованы от магнитофонов, предназначенных для больших ЭВМ. Эти магнитофоны представляют

собой большие устройства, разработанные для чтения с магнитной ленты, движущейся с большой скоростью, и записи на нее. Они имеют мощные двигатели для пуска и останова тяжелых бобин с лентой. Лента на бобине около фута в диаметре (30 см) весит два фунта (~1 кг); информация на ленте записана на девяти параллельных дорожках с плотностью 1600 бит/дюйм (64 бит/мм). Поэтому на одном дюйме магнитной ленты можно хранить 1600 байт. Данные всегда записываются блоками размером до 4К байт с зазорами в 0,5 дюйма (12,5 мм) между ними. Благодаря наличию межблочных зазоров магнитофон может тормозить и набирать скорость между блоками, так что программе доступны такие процедуры, как "пропуск N блоков" или "возврат на N блоков". Подобные команды перешли и в Бейсик, где они используются при работе с некоторыми последовательными файлами на гибких дисках, но недоступны при работе с бытовыми магнитофонными кассетами, так как кассетные магнитофоны не обладают требуемыми аппаратными свойствами.

8.4. ПОСЛЕДОВАТЕЛЬНЫЕ ФАЙЛЫ

Последовательные файлы имеют организацию, описанную в разд. 8.2, а именно, записи в них следуют одна за другой. У файла есть заголовок, за которым следуют элементы данных и, может быть, концевая часть. Доступ к отдельным элементам данных нельзя получить, не прочитав все предшествующие данные. Отдельные элементы файла не могут быть в нем изменены; должна быть создана новая версия файла, включающая в себя все изменения. В Бейсике предусмотрены расширенные варианты операторов INPUT и PRINT, рассчитанные на работу с последовательными файлами и в основном не зависящие от носителя, используемого для хранения файла. В разд. 8.4.1 описаны общие приемы работы с файлами, а в последующих разделах — особенности конкретных носителей. Эти дополнительные детали в основном связаны с созданием файла и с привязкой файла к программе. Последнее действие часто называется открытием файла, и оба этих термина (привязка, открытие) будут использоваться для обозначения одних и тех же описываемых ниже действий.

Основные действия при работе с файлами таковы:

(а) Открытие файла, т. е. привязка файла к логическому номеру. Если такого файла нет, то в некоторых системах он будет создан, а в других потребуется вначале создать его командой CREATE или особой формой команды OPEN.

(б) Запись данных в файл, или чтение данных из файла, или добавление данных к концу файла (если Ваша система допускает такой режим) .

(в) Закрытие файла, прекращающее связь между файлом и логическим номером и в большинстве систем посылающее в файл специальную метку EOF (end of file — конец файла).

8.4.1. РАБОТА С ПОСЛЕДОВАТЕЛЬНЫМИ ФАЙЛАМИ

Между чтением последовательного файла и действиями оператора READ (см. разд. 4.4), читающего последовательный список элементов данных,

ранящихся в операторах DATA, существует немалое сходство. В случае файла вместо оператора READ используется расширенный вариант оператора INPUT, а элементы данных исчезают из поля зрения и запоминаются аналогичным образом в последовательном файле.

Чтобы обеспечить гибкость манипулирования файлами, в Бейсике используется идея логического номера, привязанного к файлу, благодаря чему во всех операторах чтения и записи, работающих с этим файлом, достаточно указывать только его логический номер. Перед началом работы с файлами или перед исполнением программы посредством одного оператора или команды (например, OPEN) осуществляется привязка существующего файла к логическому номеру, например:

```
10   REM ПРИМЕР РАБОТЫ С ФАЙЛОМ
20   OPEN 1,1,0,"INFO"
30   INPUT #1,A,B,C
```

Приведенный выше пример специфичен для конкретной системы (PET), но тем не менее иллюстрирует привязку в строке 20 файла INFO к логическому номеру 1 (первая единица после OPEN). В строке 30 показан расширенный оператор INPUT, считывающий значения переменных A, B и C из файла, привязанного к номеру 1. Детали открытия файла и его привязка к логическому номеру будут приведены в последующих разделах данной главы. Перед исполнением примеров можно обратиться к соответствующему разделу в зависимости от используемой Вами системы и носителя информации архивной памяти.

Обозначение логического номера файла

Общая форма записи:

#N: *или* #N,

где N - *целочисленное выражение, переменная или константа.*

В некоторых системах существуют ограничения для номера N, *например:* 1 <=N<=15 *для Бейсика Microsoft,* 1<=N<=255 *для персональной ЭВМ PET,* 1<=N<=6 *для ЭВМ ICL 2904*

Кроме того, для общего числа одновременно открытых файлов операционной системой устанавливается предел, обычно меньший 10.

замечание. Бейсик BBC отличается тем, что значение N *не выбирается программистом, а предоставляется системой при привязке файла к программе.*

Приведенные ниже операторы могут применяться для работы с последова-

тельным файлом, если добавлять в них обозначение логического номера файла:

INPUT #N, LINPUT #N, LINEINPUT #N, GET #N, PRINT #N, PRINT #N, USING

Приведем пример программы, действия которой состоят из двух этапов. На первом происходит запись в простой файл, а на втором — считывание данных обратно в память и изображение их на ВТУ. Так как Вы не можете "видеть", что именно запомнено в файле на ленте или диске, то надо иметь средства проверки его содержимого, что и обеспечивается второй частью программы.

```
10 REM СОЗДАНИЕ И ПРОСМОТР ФАЙЛА
20 REM **ЗАМЕНИТЕ ЭТОТ КОММЕНТАРИЙ НА ОПЕРАТОР OPEN**
30 FOR I=1 TO 50
40   PRINT «2,I*I
50 NEXT I
```

```
60 CLOSE 2
70 PRINT "ФАЙЛ ЗАПИСАН"
80 PRINT
90 PRINT "ПЕРЕМОТАЙТЕ ЛЕНТУ И НАБЕРИТЕ RUN 120" 100 STOP
110 REM ПРОСМОТР ФАЙЛА
120 REM **ЗАМЕНИТЕ ЭТОТ КОММЕНТАРИЙ НА ОПЕРАТОР OPEN**
130 FOR K=1 TO 50
140   INPUT #3,X
150   PRINT X;
160 NEXT K
170 CLOSE 3
180 END
```

Строки 20 и 120 должны быть заменены на соответствующие операторы OPEN для работы с одним и тем же файлом. Для иллюстрации логический номер 2 использован при записи в файл, а 3 — при чтении из него; этот выбор совершенно произволен.

В системе BBC надо использовать операторы

```
40   PRINT #N,I*I и 140   INPUT #N,X
```

где N берется из операторов OPEN (см. подразд. 8.4.4). Так как в системе BBC в операторе RUN номер строки указать нельзя, то приведенную выше программу лучше всего разбить на две: одну составить из строк 10—100, а другую — из строк 110—180.

Действие первой части программы состоит в записи квадратов чисел от 1 до 50 в файл. Каждый оператор PRINT добавляет в файл новую запись (зрительно ее можно представлять в виде строки). По команде RUN программа

начнет выполнять указанные действия и остановится, выдав приведенное ниже сообщение. В команде RUN можно указать номер строки, с которой должно начаться исполнение программы, что и делается в нашем случае для исполнения второй части программы:

```
RUN
ФАЙЛ ЗАПИСАН
ПЕРЕМОТАЙТЕ ЛЕНТУ И ДАЙТЕ КОМАНДУ RUN 120
RUN 120
1   4   9   16   25   36   49   64   81   100   121   144
169 (и т. д.)
```

Вы можете заметить небольшую паузу в процессе вывода после изображения части чисел на экране ВТУ. Она вызывается буферизацией, которая в том или ином виде применяется при работе со всеми файлами данных. Во всех системах для каждого устройства вывода в памяти выделяется буферная область. У микроЭВМ размер этой области обычно составляет 128 или 256 байт. Данные, которые должны быть записаны в файл, собираются в этой буферной области до ее заполнения, затем все содержание буфера целым блоком передается устройству. После этого буфер готов принять новые данные и т. д. Аналогичным образом при вводе каждый физический блок, считываемый в ЭВМ, попадает в буфер; программа читает данные из этого буфера до тех пор, пока он не станет пустым, после чего с ленты или диска считывается новый блок. Упомянутая выше пауза свидетельствует о том, что программа ждет, пока буфер не заполнится новыми данными, считываемыми из внешней памяти на ленте или диске. Один из способов избежать подобной неравномерности действий — применение двух буферов таким образом, чтобы один из них заполнялся данными в то время, когда из другого извлекаются данные, и наоборот. Однако такой режим средствами Бейсика осуществить нельзя.

Модифицируем предыдущую текстовую программу так, чтобы при выводе и чтении в каждой записи файла было по два значения:

```
10 REM РАБОТА С ФАЙЛОМ - ПРИМЕР 2
20 КЕМ ««««ЗАМЕНИТЕ ЭТОТ КОММЕНТАРИЙ НА ОПЕРАТОР OPEN***
30 FOR I=1 то 50
40   PRINT #2,1;",";100*I
50 NEXT I
60   CLOSE 2
70   PRINT "ФАЙЛ ЗАПИСАН"
80   PRINT
90   PRINT "ПЕРЕМОТАЙТЕ ЛЕНТУ И НАБЕРИТЕ RUN 120"
100 STOP
110 REM ПРОСМОТР ФАЙЛА
120 REM ***ЗАМЕНИТЕ ЭТОТ КОММЕНТАРИЙ НА ОПЕРАТОР OPEN***
130 FOR K=1 TO 50
140   INPUT #3,X,Y
150   PRINT X,Y
l60 NEXT K
170 CLOSE 3
180 END
251
```

RUN
ФАЙЛ ЗАПИСАН
ПЕРЕМОТАЙТЕ ЛЕНТУ И НАБЕРИТЕ RON 120
RUN 120

```
1            100
2            200
3            300
4            400 (и т. д.)
```

Результаты исполнения соответствуют ожидаемым, так как в строке 40 выводятся два значения, I и 100*I, а в строке 140 они считываются обратно. Вернемся к строке 40 и обратим внимание на то, что в файл посылается также значение ",". Это делается потому, что операторы INPUT# и PRINT# фактически работают так же, как и исходные версии INPUT и PRINT. Это означает, что оператор INPUT ожидает в одной строке (или записи) получить все значения, разделенные запятыми. Например, оператор

10 INPUT SUM, TOT, E ожидает получения ввода с клавиатуры в форме

? 10,-6,42

Тот же относится и к файлам. Если запятые отсутствуют, то большинство систем с Бейсиком игнорирует промежуточные пробелы, в результате чего числа сливаются. Попробуйте получить подобный эффект, удаляя запятую из строки 40 и используя оператор

40 PRINT #2,I; 100*I Запустив программу вновь, Вы получите

RUN
ФАЙЛ ЗАПИСАН
ПЕРЕМОТАЙТЕ ЛЕНТУ И ДАЙТЕ КОМАНДУ RUN 120
RUN 120

```
1100      2200 3300      4400
(и т. д.) ERROR AT LINE 140
```

Сообщение об ошибке вызвано тем, что из-за отсутствия запятой программ считает, что в каждой записи содержится только один элемент данных, та что строка 140 считывает одну запись в X, а другую в Y и после выполнения половины общего числа проходов цикла пытается считать данные за концом файла.
252

Важно иметь возможность обнаруживать конец файла, так как на практике точное число элементов данных в файле не известно. В некоторых системах для обнаружения конца файла предусмотрена функция EOF ( ) :
Бейсик Microsoft: EOF(N) = -1 (ИСТИНА), если достигнут конец файла
Бейсик BBC: EOF # (N) = -1 (ИСТИНА), если достигнут конец файла
В других системах для этого предусмотрен специальный оператор IF:
ICL 2904: IF END# 1 GOTO 200
В некоторых системах имеется специальное слово состояния, значение которого зависит от реакции системы на каждый обмен данными. В персональной ЭВМ PET имеется слово состояния ST (размером в 1 байт) , и при различных реакциях в этом слове устанавливаются (т. е. принимают значение 1) разные биты:
PET: СЛОВО СОСТОЯНИЯ ST = 0, если все в порядке
= 64, если от кассетного магнитофона получен сигнал конца файла

```
100  REM ФРАГМЕНТ РАБОТЫ С ФАЙЛОМ ДЛЯ ЭВМ PET
110  INPUT #2, A,B,C
120  IF ST=0 THEN 200
130  IF ST=64 THEN 500
```

Переход к строке 200 отвечает нормальной обработке, а к строке 500 — действиям, выполняемым при обнаружении конца файла. Слово состояния имеет то преимущество, что в нем можно отмечать целый спектр возможных реакций и тем самым делать их доступными программе на Бейсике.
Используя функцию Бейсика BBC, можно привести пример обнаружения конца файла, содержащего список фамилий и чисел. Здесь был бы нужен цикл DO WHILE NOT EOF#N, но в Бейсике BBC его нет.

```
100 REM ЧТЕНИЕ ДАННЫХ ИЗ ФАЙЛА BBC С ЛОГИЧЕСКИМ НОМЕРОМ N
110 IF EOF#(N) THEN
120 INPUT#N , NAME$ , NUMB
. Процесс ввода
200  GOTO  110
210  REM ВЫХОД ПРИ ОБНАРУЖЕНИИ КОНЦА ФАЙЛА
220  CLOSE#N
```

Обратите внимание на некоторое отличие оператора CLOSE в Бейсике BBC от обычно используемой формы.
8.4.2. ФАЙЛЫ НА МАГНИТНОЙ КАССЕТЕ
Поскольку процесс передачи блоков является прерывистым, в отличие от непрерывного чтения или записи при работе с программными файлами, то
253

используемый для работы с файлами данных магнитофон должен управляться ЭВМ. Для этого ЭВМ, как минимум, должна иметь возможность включать и выключать двигатель лентопротяжного устройства после ручного включения магнитофона на чтение или запись. Хорошо, если при этом

ЭВМ умеет разбираться в том, какая из клавиш была нажата (если была), но лучше всего, если ЭВМ обладает возможностью полного управления чтением, записью и перемоткой ленты.
Обычная процедура работы с файлом такова: когда после запуска программы достигается оператор OPEN (или аналогичный ему), то система должна затребовать включение магнитофона. Под управлением ЭВМ двигатель лентопротяжного устройства должен включаться и выключаться при каждой передаче данных, и так до тех пор, пока, наконец, не будет достигнут оператор CLOSE. Ниже на примере двух систем проиллюстрированы общие свойства процесса работы с файлами данных на кассете.

8.4.3. ПЕРСОНАЛЬНАЯ ЭВМ PET ФИРМЫ COMMODORE

В ЭВМ PET применяется довольно универсальная система для коммуникации со всеми устройствами, рассчитанными на подсоединение к шине данных, отвечающих стандарту IEEE 488. Она имеет то преимущество, что пригодна для работы с множеством типов внешних устройств, но вводит некоторые дополнительные параметры в оператор OPEN, которые при работе с простыми устройствами вовсе не обязательны. В этой системе все внешние устройства имеют фиксированные (в аппаратуре ЭВМ) номера устройств от 0 до 30, некоторые из которых показаны в табл. 8.4.

Таблица 8.4. *Номера и типы устройств персональной ЭВМ PET фирмы Commodore*

Номер устройства	Устройство
0	
1	Кассета 1
2	Кассета *2*
3	Экран ВТУ
4	Принтер
8	Дисковод

Номер устройства связывается с логическим номером файла, используемым программой в операторе OPEN, общая форма записи которого такова: OPEN логический номер файла, номер устройства, вторичный адрес, имя файла
В контексте операций с лентой термин "вторичный адрес" немного сбивает с толку, так как на деле этот параметр используется для указания режима открытия устройства. По умолчанию
принимается значение 0 , отвечающее
открытию только для чтения; 1 соответствует открытию для записи, а 2 — для записи, при которой после закрытия в файл принудительно записывается метка конца файла. Имя файла указывать не обязательно, но желательно. Оно может включать в себя до 128 символов. Например, оператор
OPEN 1,2,1, "DATA FOR PAYROLL"
открывает второй кассетный магнитофон для записи на него файла DATA FOR PAYROLL и привязывает его к логическому номеру 1.

Ниже приведены операторы, которыми можно пользоваться для работы с кассетными магнитофонами:
OPEN #N, D, вторичный адрес, "имя файла" PRINT #N, INPUT #N, GET #N, CLOSE N
Размер буфера составляет 192 символа, но длина записи, используемой в одном операторе INPUT или PRINT, не должна превышать 80 символов. Не забывайте вставлять запятую "," или CHR$ (44), между значениями, входящими в состав одной записи.

8.4.4. СИСТЕМА BBC

Одна из операционных систем для микроЭВМ BBC специально рассчитана на работу с кассетой. Она называется CFS (cassette filing system — кассетная файловая система). Виды команд в этой системе и в дисковой операционной системе ничем не отличаются, но из-за особенностей, присущих ленточному носителю, возможности этих команд в кассетной системе несколько ограничены. Они являются функциями, возвращающими логический номер файла, устанавливаемый системой. Например, оператор

N=OPENIN ("имя файла")

откроет файл только для чтения и привяжет его к программе, поместив логический номер файла в переменную N:

N=OPENIN ("PROCESS") А оператор

N=OPENOUT ("имя файла") создаст новый файл и откроет его только для записи:

N=OPENOUT ("RESULTS")

Ниже приведены операторы, которыми можно пользоваться при работе с кассетой:

M=OPENIN ("имя файла")

M=OPENOUT ("имя файла")

INPUT #M,

X=BGET#(M)

PRINT #M,

BPUT #M,

CLOSE #M

Для обнаружения конца файла с логическим номером М используется функция EOF#(M). Она возвращает —1, если обнаружен конец файла; в противном случае 0. (Учтите, что в системе BBC значению "истина" соответствует -1,а"ложь"0.)

В кассетой файловой системе имена файлов не должны состоять более чем из 10 символов, но могут включать в себя специальные символы (в отличие от имен переменных).

Приведенный ниже фрагмент программы иллюстрирует применение некоторых из указанных выше операторов. Он запоминает список адресов в файле по имени MAILLIST. Адреса вводятся с клавиатуры в строке 140 в строковую переменную ADDS и записываются из нее в файл в строке 150.

```
100 REM СОЗДАНИЕ ПОСЛЕДОВАТЕЛЬНОГО ФАЙЛА В БЕЙСИКЕ BBC
110 N=OPENOUT("MAILLIST")
120 INPUT "ВВЕДИТЕ ЧИСЛО АДРЕСОВ".M
130 FOR I=1 TO M
140 INPUT "АДРЕС".ADD$
150       PRINT#N.ADD$
160 NEXT I
170 CLOSE#N
```

Оператор OPENIN открывает существующий кассетный файл для ввода, а оператор OPENOUT открывает файл для вывода из программы. Как INPUT#, так и PRINT# могут обрабатывать все типы переменных и обеспечивать обмен и числовыми значениями, и строками символов между программой и файлом. В дополнение к этому операторы BGET# и BPUT# дают возможность передачи отдельных байтов из файла в программу и наоборот. Приведенная ниже программа использует BPUT для запоминания отдельных байтов в кассетном файле:

```
10 REM СОЗДАНИЕ ПОСЛЕДОВАТЕЛЬНОГО ФАЙЛА В БЕЙСИКЕ BBC
20 L=OPENOUT("CHRS")
30 REPEAT
40   A$=GET$
```

```
50   BPUT#L,ASC(A$)
60 UNTIL A$="*"
70 CLOSE#L
80 END
```

Байты образуются за счет ввода с клавиатуры отдельных символов с помощью GETS в строке 40, а цикл REPEAT завершается при нажатии на клавишу со звездочкой (*).

8.4.5. ФАЙЛЫ НА ГИБКИХ ДИСКАХ

Этот подраздел посвящен последовательным файлам на гибких дисках; по поводу деталей работы с файлами прямого доступа обратитесь к разд. 8.5, хотя многие из операторов, требуемых для работы с дисковыми файлами прямого доступа, представлены в настоящем подразделе. В следующих подразделах обсуждаются дисковые файловые системы для персональных ЭВМ PET фирмы Commodore, BBC, а также для Бейсика Microsoft при работе с операционной системой CP/M.

8.4.6. ДИСК ПЕРСОНАЛЬНОЙ ЭВМ PET ФИРМЫ COMMODORE

Для обмена данными как с диском, так и с лентой ЭВМ PET использует шину данных, соответствующую стандарту IEEE 488. Оператор OPEN имеет

ту же форму, что и в случае ленты, но имя файла должно нести дополнительную информацию, а вторичный адрес становится физическим каналом диска со специальными свойствами, показанными в табл. 8.5.

Таблица 8.5. *Значения дискового вторичного адреса для персональной ЭВМ PET*

Дисковый вторичный адрес	Значение
0	Используется командой LOAD
1	Используется командой SAVE
2-14	Ввод-вывод данных (до пяти номеров может быть использовано параллельно)
15	Канал команд

Команды передаются по каналу, задаваемому вторичным адресом 15, оператором PRINT, а реакция на эти команды воспринимается оператором INPUT. Этот канал имеет свой оператор OPEN, который должен быть выполнен до открытия любых каналов для обмена данными. Например, операторы

```
10   OPEN 15,8, 15 20   PRINT #15, "10"
```

используют логический номер 15 (первый параметр) для привязки к дисковому устройству (устройство 8) с вторичным адресом 15 (канал команд) . Для инициации работы с диском, установленным в дисководе 0, команды посылаются в виде строки символов. Приведем перечень различных команд:

N — обновление,
I — инициация,
D — копирование диска,
C — копирование файла,
R — удаление файла,
S — объявление файла рабочим,
$ — запрашивание справочника файлов (см. ниже) .

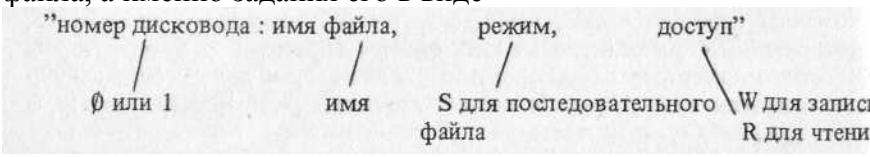
За полными сведениями обратитесь к руководству по дисковой операционной системе (ДОС) ЭВМ PET. Реакция на команды возвращается в специальном формате и может быть получена оператором

```
INPUT #15, EN, EM$, ET, ES
```

(логический номер 15 выбран совпадающим с каналом команд для удобства запоминания), где
EN содержит 0, если команда выполнена, или номер ошибки в противном

случае;
EM$ содержит сообщение об ошибке; ET содержит номер дорожки; ES содержит номер сектора.
257

В ДОС имеется справочник файлов, который (при условии, что диск находится в дисководе 0) может быть получен следующим образом:
OPEN 15,8,15 (открыть диск и канал команд) ;
LOAD "$0", 8 (запрос на загрузку справочника файлов);
LIST (изображение справочника файлов).
Программы используют команды LOAD и SAVE, например:
LOAD "0: PROCESS", 8 SAVE "0: RESULTS", 8
где 0 означает дисковод 0. Перейдем, наконец, к команде OPEN для открытия последовательных файлов, требующей расширения параметра, стоящего в позиции имени предыдущего (кассетного) файла, а именно задания его в виде



Номера дисковода соответствуют двум дисководам, встроенным в один корпус. Таким образом, для открытия файла данных можно использовать оператор
OPEN 4, 8,10, "0:PROCESS DATA, S, R"
Он открывает для чтения файл PROCESS DATA, находящийся на диске, номер которого задается вторичным адресом 10, и привязывает его в программе к логическому номеру 4. После открытия программа может использовать операторы
INPUT #4 PRINT #4 GET #4 CLOSE #4
Общая процедура работы с последовательным файлом в системе PET может быть описана следующим образом:
(а) В самом начале работы программы надо инициировать диск, на котором находятся данные, и открыть канал команд вне зависимости от того, надо ли читать данные из файла или записывать в него:
10 OPEN 15, 8, 15
20 PRINT #15, "10"
30 GOSUB 1000
Оператор GOSUB вызывает подпрограмму, обрабатывающую реакцию системы.
(б) Затем надо открыть файл либо для записи (если одноименный файл уже существует, то он будет уничтожен) посредством оператора
258

50 OPEN CH, 8, CH, "@0: INFO, S, W"
где CH содержит логический номер файла, либо для чтения посредством оператора
50 OPEN CH, 8, CH, "0: INFO, S, R"
После каждого такого оператора надо обратиться к подпрограмме обработки реакции системы. При чтении проверьте слово состояния ST, которое должно быть равно 0 при нормальном чтении и 64 при обнаружении конца файла.
(в) После этого можно осуществлять считывание или запись с помощью операторов INPUT#, GET# или PRINT#. После каждой операции надо обращаться к подпрограмме обработки реакции системы.

(г) Перед концом работы программы надо закрыть файл. Подпрограмма обработки ошибок должна иметь следующий вид:

```
500   REM ПОДПРОГРАММА ОБРАБОТКИ ОШИБОК
510   INPUT #15, EN, EM$,ET, ES
520   IF EN=0 THEN RETURN
530   REM НАПЕЧАТАТЬ СООБЩЕНИЕ ОБ ОШИБКЕ
540   REM И ЗАКОНЧИТЬ РАБОТУ ЛИБО ЗАПРОСИТЬ
550   REM С ТЕРМИНАЛА ДЕЙСТВИЕ И ВЕРНУТЬСЯ В ПРОГРАММУ
```

8.4.7. ДИСКОВАЯ СИСТЕМА BBC

Для работы с гибкими дисками система BBC использует дисковую файловую систему. В части программирования работа с последовательным дисковым файлом очень похожа на описанную ранее работу с кассетным файлом. Операторами OPENIN и OPENOUT можно пользоваться так же, как и ранее, но теперь OPENIN позволяет выполнять как чтение, так и прямой доступ к файлу, а OPENOUT — последовательную запись в файл.

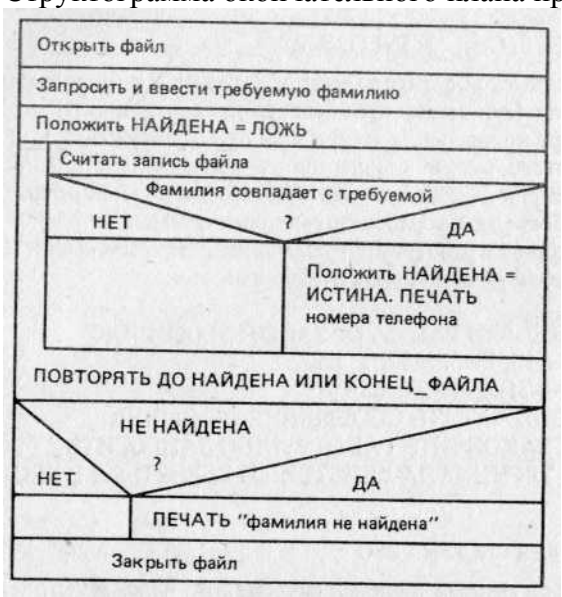
Каких-либо специальных параметров, позволяющих отличить последовательные дисковые файлы от файлов прямого доступа, не существует, однако обратитесь к комментарию в подразд. 8.6.3 по поводу дальнейшего развития системы. При работе с последовательными дисковыми файлами можно пользоваться следующими операторами:

```
M=OPENIN M=OPENOUT INPUT #M, PRINT #M, X=BGET#(M) BPUT #M, CLOSE #M
```

Пусть последовательный файл TELEPHONE состоит из записей, содержащих фамилии и телефонные номера. Приведенная ниже программа должна

просмотреть файл в поисках фамилии, совпадающей с введенной, и при обнаружении изобразить номер телефона.

Структограмма окончательного плана программы такова:



Ее воплощением служит следующая программа:

```
10 REM ДЕМОНСТРАЦИЯ РАБОТЫ С ПРОСТЫМ ФАЙЛОМ
20 M=OPENIN("TELEPHONE")
30 INPUT "ВВЕДИТЕ ТРЕБУЕМУЮ ФАМИЛИЮ, ПОЖАЛУЙСТА";QUERY$
40 FOUND=8                :REM ЛОЖЬ
50 REPEAT
60   INPUT#M.NAM$,NUMB$
```

```
70   IF NAM$<>QUERY$ THEN 100
80   FOUND=-1       :REM ИСТИНА
90   PRINT "НОМЕР ТЕЛЕФОНА: ";NUMB$
100 UNTIL (EOF#M OR FOUND)
110 IF NOT FOUND THEN PRINT "ФАМИЛИЯ НЕ НАЙДЕНА"
120 CLOSE#M
```

8.4.8. ФАЙЛЫ В БЕЙСИКЕ MICROSOFT

В данном подразделе обсуждается работа средствами Бейсика Microsoft в операционной системе CP/M с последовательными файлами на гибком диске. В подразд. 8.5.2 подробно обсуждается система CP/M.

Первый параметр в операторе OPEN обозначает режим доступа к файлу, и для последовательного чтения он должен быть равен I, а для последовательной записи O. Следующие два параметра — логический номер файла (от 1 до 15) и имя файла. Длина имени файла может составлять до восьми симво-

лов; кроме того, допускается его расширение до трех символов, отделяемое от первой части точкой. Примерами допустимых в CP/M имен файла служат

FIRST.DAT DATA. PAY DATA. SAL

Оператор OPEN вида

```
10   OPEN "I", #3, "DATA.PAY"
```

открывает программе файл DATA.PAY для ввода и привязывает его к логическому номеру 3. При работе с последовательными дисковыми файлами можно пользоваться операторами

OPEN
INPUT#
LINEINPUT#
PRINT#
PRINT#USING
WRITE#
CLOSE

Особенно полезен оператор WRITE#, поскольку он выводит данные точно так же, как оператор PRINT, но вдобавок вставляет запятые для разделения всех значений. Таким образом, операторы

```
100  PRINT #3, A;","; B;","; C
```

и

```
100  WRITE #3, A, B, C
```

равносильны и требуются в том случае, если файл впоследствии будет считываться программой с помощью оператора INPUT#.

Для обнаружения конца файла с логическим номером N используется функция EOF (N), а функция LOC, применяемая при работе с файлами прямого доступа, возвращает число секторов по 128 байт, считанных или записанных в последовательном режиме с момента открытия файла.

Режим добавления данных к файлу отсутствует. Если существующий файл открывается для записи (с режимом доступа O), то его первоначальное содержание уничтожается.

8.5. ФАЙЛЫ В ТЕРМИНАЛЬНОМ ФОРМАТЕ

Некоторые системы позволяют распечатывать на ВТУ последовательные файлы, созданные так, как описано в предыдущих частях разд. 8.4, а также редактировать их подобно программам на Бейсике. В действительности, информация хранится в этих файлах в том же виде, как если бы она была введена непосредственно с терминала ЭВМ.

Подобные файлы довольно обычны для больших ЭВМ и хранятся в них на больших дисковых системах. Примеры работы с файлами этого типа описаны ниже для системы ICL 2904.
261

МикроЭВМ, вообще говоря, не обладают достаточно хорошей операционной системой, обеспечивающей возможность работы с подобными файлами в терминальном формате. Исключением являются микроЭВМ с операционной системой CP/M. Они описаны в подразд. 8.5.2 вместе с кратким перечнем общих возможностей манипулирования файлами в CP/M.
Подобные возможности обеспечивает и дисковая система для микроЭВМ BBC. Команда *BUILD позволяет создавать текстовые файлы непосредственно с клавиатуры. Эти файлы могут быть изображены на ВТУ командами *LIST и *TYPE.

8.5.1. ФАЙЛЫ В СИСТЕМЕ ICL 2904

В этой системе предусмотрены файлы двух типов: в терминальном и внутреннем форматах. С последним можно работать в режиме прямого доступа (см. подразд. 8.6.1).
Последовательные файлы в терминальном формате можно создавать командой OPEN, в отличие от многих других систем не являющейся оператором. Чтение из файла и запись в файл осуществляются программами обычным образом. В приведенном ниже перечне указаны предоставляемые системой средства.
Только команда:
OPEN имя файла, размер в записях (резервируется на диске место для файла) Операторы:

```
10   FILE #N: "имя файла"     (привязывает файл к логическому номеру)
10   INPUT #N:
10   LINPUT#N:
10   PRINT #N:
10   FILE#N:"*"                (закрывает файл)
```

Длина имени файла в этой системе ограничена шестью символами. Для проверки достижения конца файла предусмотрены два оператора IF:

```
IF END #N THEN 100
IF MORE #N GOTO 10
```

Очень важным и полезным свойством является возможность возвращения к началу файла (по аналогии с RESTORE в операторах DATA) посредством оператора RESET #N. Его функции этим не ограничиваются; если в нем вторым параметром указан номер записи, то он переместит текущую позицию чтения или записи в файле на запись с указанным номером, обеспечив тем самым возможность прямого доступа к информации:

```
100   RESET #3 100   RESET #3,45
```

Файловая функция LOC(#N) возвращает номер текущей записи, а полезная функция TYP (#N) анализирует следующий за текущим элемент данных файла для определения его типа (1 — числовое значение, 2 — строка символов, 3 — конец записи и 4 — конец файла).
262

Файлы можно использовать и по-другому, манипулируя ими непосредственно с клавиатуры. В описываемой системе манипулирование программами и файлами осуществляется единым образом, так что обычные команды GET, LIST и команды редактирования строки применимы и к файловым данным. В дополнение к ним предусмотрена специальная команда, TEXT, создающая новые файлы

данных. В приведенном ниже примере то, что набирается на клавиатуре, подчеркнуто; система в качестве приглашения к вводу использует знак >, но при вводе данных в файл меняет его на <:

>TEXT INFO (Надо создать новый файл с именем INFO)
<ПЕРВАЯ СТРОКА,2.3 <4.678, ВТОРАЯ,7.8 < 1 . 2 . 3.
<ПОСЛЕДНЯЯ, 67
<//// (Четыре / завершают ввод)
INFO SAVED
>LIST (Изображение содержимого текущего Файла) INFO
1 ПЕРВАЯ СТРОКА, 2.3
2 4.678, ВТОРАЯ, 7.8
3 1,2,3
4 ПОСЛЕДНЯЯ, 67 >

Можно пользоваться следующими командами:

LIST изображение на ВТУ текущего программного файла или файла данных;
KILL INFO удаление поименованного файла (INFO) из архивной памяти;
SCRATCH удаление текущего файла из рабочей области памяти ЭВМ;
GET INFO копирование файла из архивной памяти в рабочую область памяти.

Программные файлы или файлы данных, находящиеся в рабочей области, можно изменять, задавая номер строки, за которым следует ее новое значение. Указание одного только номера строки вызывает удаление строки, а в результате исполнения оператора

DELETE 10,50

будут удалены строки с 10-й по 50-ю включительно. Чтобы возвратить измененную копию файла в архивную память, надо удалить исходный файл по команде KILL, после чего текущий файл можно скопировать из рабочей области по команде SAVE. Имя находящегося в рабочей области файла можно изменить по команде NAME.

Следующая простая программа использует созданный выше файл INFO. Так как он содержит смесь строк символов и чисел, то все его элементы читаются как строки символов. Запятая в операторе INPUT после A$ означает, что оператор должен продолжить чтение следующего элемента данных из

текущей записи; без этой запятой программа читала бы только первые элементы каждой записи.

```
10  REM ДЕМОНСТРАЦИЯ РАБОТЫ С ФАЙЛАМИ ЭВМ ICL 2904
20  FILE#l: "INFO"
30  INPUT#1: A$,
40  PRINT "ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ: ";A$
50  IF MORE#1 THEN 30
60  FILE#l: "*"
70  END
RUN
ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ:  ПЕРВАЯ СТРОКА
ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ:  2.3
ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ:  4.678
ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ:  ВТОРАЯ
ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ:  7.8
ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ:   1
```

ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ: 2
ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ: 3
ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ: ПОСЛЕДНЯЯ
ОЧЕРЕДНОЙ ЭЛЕМЕНТ ДАННЫХ: 67 LINE 70 DONE

8.5.2. ФАЙЛЫ В ОПЕРАЦИОННОЙ СИСТЕМЕ CP/M

В операционной системе CP/M физические дисководы имеют метки A, B, C и т. д. По соглашению метку A имеет дисковод, встроенный в корпус ВТУ. Системный диск, содержащий CP/M, должен быть вставлен в дисковод A. После того как CP/M будет загружена в память посредством описанной в подразд. 2.1.2 процедуры, приглашение к вводу будет содержать указание на текущий дисковод:

A>

Набрав B: или C:, можно изменить текущий дисковод на B или C соответственно.

Учтите, что при загрузке Бейсика командой MBASIC по умолчанию разрешается работать не более чем с тремя файлами данных, которые должны иметь логические номера от 1 до 3. При необходимости в работе с большим числом файлов надо указать в команде параметр"/F: число файлов". Например, команда

A>MBASIC/F:6

позволит Бейсику использовать до шести файлов данных. Все непосредственные манипуляции файлами, включая их создание, выполняются вне Бейсика. Ниже будут использованы ссылки на команды CP/M и ее редактор.

При работе с файлами часто бывает необходимо указывать, на каком дисководе (A,B,C) располагается файл. Некоторые команды CP/M способны манипулировать группами файлов и могут использовать звездочку (*) для обозначения "любой строки символов" в имени файла, например:

A:FIRST.PRG означает файл FIRST.PRG на дисководе A;
SAMPLE.TXT означает файл SAMPLE.TXT на текущем дисководе;

B:*.TXT означает все файлы на дисководе B, имеющие расширение имени TXT;
B:*.* означает все файлы на дисководе B.

учтите, что команды должны вызываться с того диска, на котором они находятся (обычно с диска A). Если это не текущий диск, то перед командой
надо набрать A:.

Последовательные файлы можно создавать обычным образом из программы, как описано в подразд. 8.4.8, или же непосредственно с клавиатуры. В последнем случае используйте команду редактирования ED операционной системы CP/M, указывая в ней имя нового файла. Данные можно ввести в файл с помощью команды редактора I; признаком конца ввода данных служит нажатие клавиши Z при нажатой клавише управляющего регистра CONTROL. Ниже приведен пример полного сеанса редактирования (набираемая на клавиатуре информация подчеркнута) :

B><u>A: ED INFO.DAT</u> (вызвать команду ED с дисковода A для создания поименованного файла на текущем дисководе, т. е. B) NEW FILE (на диске B нет файла с указанным выше именем,
поэтому система создает новый файл)
:*<u>I</u> (редактор выдает звездочку в качестве приглашения к вводу; для перехода в режим вставки текста наберите I)
1:<u>10</u>
2:<u>40</u>
3:<u>75</u> (вводите значения данных, нажимая клавишу воз-
4:<u>-б</u> врата каретки после каждой набранной строки)
5:<u>21</u>

6: (наберите CONTROL Z для завершения исполнения команды I)
:*E (снова выдано приглашение к вводу (*); введите
E для завершения редактирования)
B> (произошел возврат к CP/M; можно вводить новую команду операционной системы).
Созданный таким образом файл INFO .DAT можно изобразить на ВТУ командой
B>A:TYPE INFO.DAT или, если текущим является диск A,
A>TYPE B:INFO.DAT
Теперь можно войти в Бейсик и обеспечить доступ программы к файлу с помощью оператора OPEN так, как это было описано выше. У Бейсика нет

команды для распечатывания или редактирования файла данных; встроенный в него редактор рассчитан только на программы (см. табл. 8.6) и никак не связан с описанным выше редактором операционной системы CP/M, В табл. 8.6 приведен список дополнительных команд, которые можно давать і в Бейсике, работающем под управлением операционной системы CP/M,

Таблица 8.6. *Дополнительные команды в Бейсике, работающем под управлением операционной системы CP/M*

Пример команды Бейсика	Действие
FILES "B:*.*"	Изображает список файлов диска B
LOAD "B:MYPROG""	Загружает программу MYPROG .BAS с диска B
SAVE "B: MYPROG"	Сохраняет программу MYPROG. BAS на диске B
SYSTEM	Завершает работу Бейсика и возвращает управление операционной системе CP/M. Если текущая программа не была сохранена на диске, то она будет потеряна
EDIT 35	Позволяет начать редактирование строки 35 текущей программы. Некоторые из команд редактирования таковы: L - печать текущей строки; пробел - перемещение курсора к следующему символу; D — удаление символа; C — замена текущего символа строки на следующий набираемый символ; I - вставка символов (завершается нажатием на клавишу ESC); возврат каретки — нажатие этой клавиши завершает редактирование и вызывает запоминание измененной строки

Выйдите из Бейсика в CP/M с помощью команды SYSTEM и войдите в него снова с помощью команды MBASIC. Итак, для работы с файлами операционной системы CP/M можно использовать какую-либо из следующих двух процедур:

Загрузите ОС CP/M и Бейсик. Создайте файлы и работайте с ними программными средствами так, как это описано в подразд. 8.4.1 и 8.4.8. Выйдите из Бейсика, если потребовалось изменить или распечатать эти файлы, либо

Загрузите CP/M и создайте или распечатайте требуемые файлы. Затем войдите в Бейсик и загрузите в его память программы, использующие эти файлы. Выйдите из Бейсика, если потребуется изменить или распечатать эти файлы непосредственно с клавиатуры.

В заключение подраздела вкратце опишем общие операции для манипулирования файлами в операционной системе CP/M. Начнем с обзора возможностей редактора.
Команда редактирования ED имеет формат
ED имя_файла
В качестве приглашения к вводу редактор выдает звездочку (*). После ее появления можно дать одну из приведенных ниже команд редактирования. Далее через ↑Z обозначен набор Z при нажатой клавише CONTROL, а + или -обозначает перемещение к началу или концу файла. Для завершения редактирования вводится либо команда E, обеспечивающая нормальный конец работы и сохранение редакции в новом файле, либо Q, обеспечивающая выход из редактора без сохранения редакции в файле. При внесении изменений в уже существующий файл после выхода из редактора исходная версия файла сохраняется на диске под именем "имя_файла.BAK". Ниже приводится перечень команд редактора:

nA	Добавить n строк
±B	Перейти к концу (+) или началу (—) буфера.
±nC	Передвинуть текущую позицию редактирования на ± n символов.
±nD	Удалить n символов.
E	Завершить редактирование и закрыть файлы.
nF	Найти n-е вхождение образца: F образец ↑Z.
H	Завершить редактирование и оставить файлы открытыми.
I	Вставить символы: I строка_символов ↑Z.
nJ	Найти образец 1 с заменой на образец 2 всех последующих символов вплоть до образца 3.
±nK	Удалить (уничтожить) n строк.
± nL	Переместиться вниз (+) или вверх (—) на n строк.
M	Макроопределение.
N	Найти образец с автоматической подкачкой строк из входного файла.
0	Возвратиться к исходному файлу.
±nP	Переместиться на n страниц с печатью.
Q	Выйти из редактора без изменения файла.
nS	Заменить образец 1 на образец 2: S образец 1 ↑Z образец 2 ↑Z.

±nT	Напечатать (изобразить) n строк.
±U	Объявить замену строчных букв на прописные (+) или отменить объявление

замены (-).

nW	Записать в выходной файл n строк.
nZ	Пропустить n строк.
± n (возврат каретки)	переместиться на n строк и напечатать новую текущую строку (эквивалентно ± n L T) .

В CP/M имеются следующие команды манипулирования файлами: команда печати справочника файлов DIR, например:

DIR
DIR B:
DIR PROG.*
DIR*.PRG

Команда изображения содержания файла на ВТУ TYPE, например:

TYPE SAMPLE.TXT
TYPE B:FRED.APG

Команда STAT, выдающая паспортные данные файла и сведения о свободной памяти, например:

STAT DSK: (информация обо всех активных дисках)
STAT
STAT B:
STAT SAMPLE.TXT
STAT*.TXT

Команда удаления файла ERA, например:

ERA SAMPLE.TXT
ERA*.TXT
ERA B:*.PRG

Команда переименования файла REN (меняет имя файла, указанного вторым, на первое имя), например:

REN NEWFL.TXT =OLDFL.PRG
RENA.B=X.Y
REN B: FRED.APG =B: JIM.BAK

Команда копирования файлов PIP, например:

PIP A:NEW.TXT=B:OLD.PRG (скопировать из OLD в NEW)

С помощью команды PIP можно скопировать файл на устройства: CON:-на консоль (ВТУ), LST:—на принтер, например: PIP CON: = SAMPLE.TXT PIP LST:=A.PRG

С помощью PIP можно копировать целиком диски, например: P1P C:=A:*.*

табл 8.7 описаны действия, выполняемые операционной системой CP/M при наборе на клавиатуре некоторых управляющих символов. Учтите, Таблица 8.7. *Действие управляющих символов в операционной системе CP/M*

управляющий символ	Действие
Общего назначения	
RUBOUT/DELETE	Удаляет символ, изображая его на экране
CTRL U или CTRL X	Удаляет строку
CTRL L	Используется после удаления для изображения "чистой" строки

CTRL E	Позволяет продолжить набор команды на следующей строке
CTRL C	Перевызов операционной системы CP/M
Прочие	
CTRL H	Возврат на одну позицию
CTRL J (переход к новой строке)	Завершает ввод
CTRL M (возврат каретки)	Завершает набор команды
CTRL P	Включает/выключает печать на принтере появляющегося на экране ВТУ изображения
CTRL S	Останавливает/возобновляет выдачу изображения на экран ВТУ

что при работе с Бейсиком эти управляющие символы могут вызвать совершенно другие действия.

8.6. ФАЙЛЫ ПРЯМОГО ДОСТУПА

Эффективность использования дисковой памяти при режиме прямого доступа обычно достигается за счет определенных ограничений на длину записи и формат запоминаемой информации. Вообще говоря, чем сложнее Дисковая операционная система, тем менее существенны эти ограничения.

Обычно файлы прямого доступа имеют фиксированный формат и фиксированную длину записи. Для гибких дисков может требоваться, чтобы длина записи совпадала с размером сектора, например 128 байт. Как числовая, так и символьная информация могут храниться либо в двоичной форме внутреннего представления данных в ЭВМ, либо полностью в символьном виде.

Объем работы, выполняемой программой, отражает степень сложности операционной системы. В основном программист требует перемещения определенных наборов значений из программы в указанные записи файла и обратно. Детали работы с буферами файлов, блоками, дорожками, размеры блоков и номера секторов, а также преобразование числовых значений не Должны входить в его обязанности. Однако в большинстве случаев, даже при

работе на больших ЭВМ, определенные познания в этой области все же оказываются необходимыми.

Следующие два раздела иллюстрируют подход, при котором формат записей файла прямого доступа определяется программистом с помощью соответствующего описания. При этом все записи файла будут иметь один и тот же формат и работа с файлами прямого доступа будет достаточно бесхитростной. Для примера выбраны две системы: большая ЭВМ ICL 2904 и Бейсик Microsoft, предназначенный для работы в операционной системе CP/M.

Системы, в которых формат записи не задается, сильно различаются, Система BBC ориентирована на побайтовую (посимвольную) передачу данных и производит впечатление достаточно удобной в работе, а система PE7 фирмы Commodore, также ориентированная на побайтовую передачу данных, требует от пользователя выполнения ряда манипуляций номерами дорожек и секторов и пользоваться ею довольно сложно.

В большинстве систем предусматривается доступ к файлу прямого доступа по номеру записи (номеру байта в Бейсике BBC). Положение записи в файл может быть связано с ее содержанием с помощью хэш-функции, вычисляющей номер записи по значению ее ключевого поля. Этот метод широко распространен в информационно-справочных системах.

8.6.1. ФАЙЛЫ ПРЯМОГО ДОСТУПА В СИСТЕМЕ ICL 2904

Для создания такого файла и описания формата записи используется команда OPEN. Все записи файла будут иметь одинаковый формат и одинаковую длину. Команда OPEN имеет следующий вид:
OPEN имя файла (формат записи), число записей Например, команда
OPEN DAFL (N, S10**,** 2N), 100

инициирует работу с файлом DAFL, содержащим 100 записей, в которых информация расположена следующим образом: "число, строка из 10 символов, два числа". Формат состоит из дескрипторов S и N:

N соответствует одному числовому значению, Sm соответствует строке из m символов.

Перед дескриптором можно указать коэффициент кратности, так что (N,N) эквивалентно (2N).

После того как файл открыт по команде OPEN, можно использовать следующие операторы:

```
10  FILE #N: "имя файла" 10  READ#N,L: 10  WRITE #N,L: 10  #N:"*"
```

Операторы прямого доступа READ и WRITE содержат номер записи (L в приведенном выше примере), указывающий, с какой именно записью они должны работать. Например, оператор

```
100  WRITE #2,1 :A,B,C$ 270
```

запишет два числовых значения и одну строку символов (при условии, что это соответствует формату записи) в запись 1 файла с логическим номером 2 Чтение информации из файла и запись ее в файл осуществляются в произвольном порядке, что позволяет выполнять любые модификации содержания записей. В принципе способом доступа к своим компонентам файл прямого доступа напоминает массив, но возможности применения файлов намного богаче, поскольку каждая запись может обладать сложной структурой, составленной из чисел и строк символов.

Номер текущей записи можно определить с помощью функции LOC (#N). Например, оператор

```
10  READ#2, LOC(#2)+1:A,B,C$
```

прочтет следующую запись файла.

Операторы IF END и IF MORE можно применять в сочетании с оператором RESET #N,L, описанным в подразд. 8.5.1.

8.6.2. ФАЙЛЫ ПРЯМОГО ДОСТУПА В БЕЙСИКЕ MICROSOFT

Хотя в этих файлах информация хранится в двоичном виде, все обмены данными между программой и файлом происходят в форме строк символов. Для облегчения преобразования числовых значений в строки символов и обратно предусмотрен набор специальных функций.

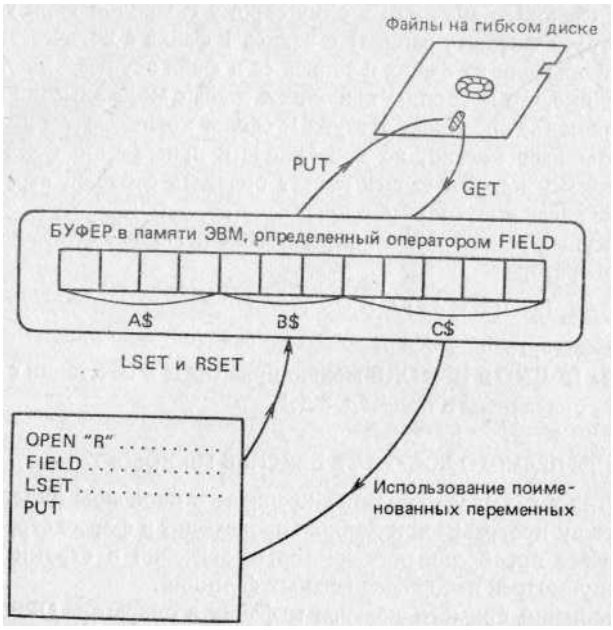
Для объявления прямого доступа к файлу в операторе OPEN в качестве первого параметра, определяющего вид доступа (см. подразд. 8.4.8), указывается R. Например, оператор

```
10  OPEN "R", #2, "DAFL", 20
```

откроет для прямого доступа файл DAFL с длиной записи 20 байт (по умолчанию 128) и привяжет к нему логический номер 2.

Дополнительная сложность работы с файлами прямого доступа состоит в том, что программа должна явным образом манипулировать данными из буфера, выделяемого системой для каждого файла. Добавочные действия связаны с необходимостью перемещения информации между буфером и программой и между буфером и файлом. Схема таких обменов изображена на рис. 8.8.

Оператор OPEN определяет длину записи (четвертый параметр), а оператор FIELD определяет буфер для файла. Длина буфера не должна превышать длину записи; все эти длины исчисляются в байтах. Кроме того, оператор FIELD определяет строковые переменные с произвольно выбираемыми именами и ассоциирует их с отдельными частями буфера. Эти переменные могут быть обычным образом использованы в операторах PRINT, строковых функциях и т. д., но непосредственное присваивание значений этим переменным не допускается. Иначе говоря, они не могут быть указаны в операторе INPUT и в левой части оператора присваивания LET. Единственным способом занесения информации в эти переменные является применение операторов LSET и RSET (см. рис. 8.8). Определение буфера посредством FIELD имеет вид



Программа на языке Бейсик Рис. 8.8. Работа с файлами прямого доступа в Бейсике Microsoft

FIELD логический номер файла, длина поля AS строковая переменная,..

где последний параметр повторяется столько раз, сколько потребуется, например:

10 FIELD #1,20 AS Z$,10 AS NAME$

Приведенный выше оператор определяет буфер для файла с логическим номером 1, содержащий 30 байт, из которых первые 20 выделяются для строки Z$, а следующие 10 — для строки NAME$. Для чтения информации из такого файла первым делом надо считать требуемую запись в буфер, а затем использовать соответствующую переменную в программе. Например, оператор

50 GET #1,209

считает запись с номером 209 из файла с логическим номером 1 и поместит ее в буфер, а операторы

60 PRINT "ФАМИЛИЯ:"; NAMES 70 PRINT "ПОДРОБНОСТИ: "; Z$

Iиспользуют значения буферных переменных, определенных оператором FIELD.

Для записи данных в рассмотренный выше файл информация должна быть помещена в буферные строковые переменные, после чего буфер должен быть переписан в требуемую запись файла. Единственным средством помещения формации в буферные переменные являются операторы LSET и RSET, например.

75 LSETZ$="99-641" + "BSI644" 80 RSET NAME$ =B$

Оба оператора выполняют одни и те же действия, только LSET выравнивает по левому, а RSET — по правому краю поля. Если при этом все поле оказалось заполненным, то недостающие позиции заполняются пробелами; если же данных в избытке, то лишние символы игнорируются. Как только данные попали в буфер, его надо переписать в требуемую запись файла, например:

90 PUT #1,304

Один оператор PUT или один оператор GET оперирует целым буфером.

Приведем перечень операторов, используемых для работы с файлом прямого доступа:

OPEN "R", # 1, (открыть файл, указать длину записи)
FIELD # 1, (определить буфер)
LSET или RSET (можно повторять много раз для записи данных
PUT#1, в файл)

GET #1, (можно повторять много раз для считывания данных из файла)
CLOSE 1 (прекращает связь между логическим номером и файлом)

Файловая функция LOC(N) возвращает номер записи, использованный в PUT или GET, увеличенный на 1, что полезно при работе с файлом в последовательном режиме.

Числа можно преобразовать в строки и получать из строк с помощью двух Функций STR$(X) и VAL(A$), обсуждавшихся в гл. 4. Однако для более быстрого преобразования чисел в строки символов можно пользоваться следующими специальными функциями:

MKI$ (I%) для целых значений, дает строку из двух байтов;
MKS$ (X) для вещественных значений, дает строку из четырех байтов;
MKD$(A#) для вещественных значений с двойной точностью, дает строку из восьми байтов.

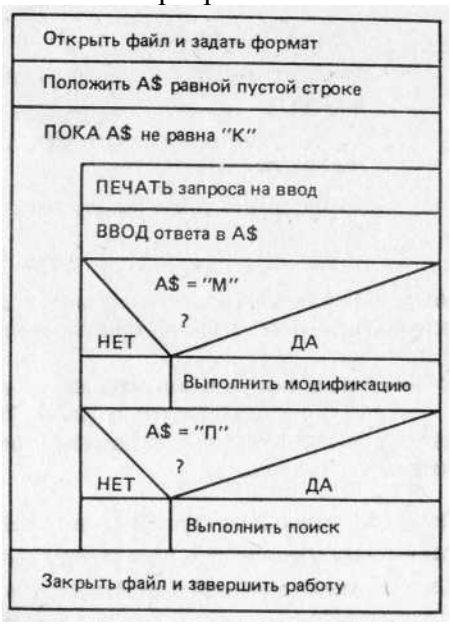
Обратные действия выполняются функциями CVI(A$), CVS(A$) нCVD(A$). Приведенная ниже программа иллюстрирует применение файлов прямого доступа в системе учета товаров. Она упрощена в целях демонстрации существенных моментов при манипулировании файлами. Пусть у лавочника на складе около 100 различных товаров, которые надо регистрировать по следующей форме:

артикул, наименование, количество на складе, цена за штуку

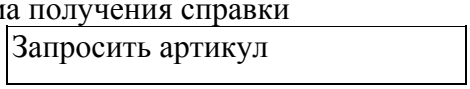
Для простоты предполагается, что лавочник ввел свою систему артикулов в виде номеров от 1 до 100, что позволяет использовать артикул как номер записи в файле, в которую помещена остальная относящаяся к нему информация.

Для ведения этой инвентаризационной ведомости нужна система, позволяющая получать справки и вносить изменения. Для ее разработки нетрудно применить структограммы. Ниже показаны три диаграммы для заключительного этапа разработки: одна для основной программы и по одной для двух подпрограмм:

Основная программа

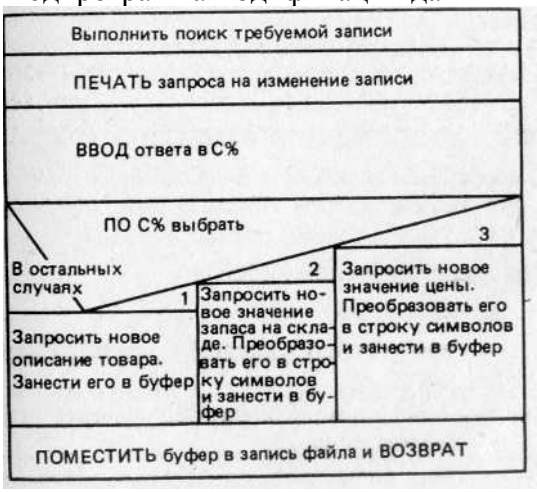


Подпрограмма получения справки



ВВОД ответа в PC%
Считать из файла запись с номером PC%
ПЕЧАТЬ содержащихся в записи данных и ВОЗВРАТ

Подпрограмма модификации данных



Законченная программа имеет следующий вид:

```
10 REM ПРИМЕР МОДИФИКАЦИИ ИНВЕНТАРИЗАЦИОННОЙ ВЕДОМОСТИ
20 OPEN "FT , #1,"STOCK",26
30 FIELD#1.20 AS DES$.2 AS SN$,4 AS P$
40 A$=""
50 WHILE A$<>"K"
60      PRINT "ВЫБЕРИТЕ:  МОДИФИКАЦИЯ.  ИЛИ  ПОИСК.  ИЛИ  КОНЕЦ
65      PRINT "                    (М.  ИЛИ  П,  ИЛИ  К)"
70      INPUT  A*
80   IF A$="M" THEN GOSUB 190
90   IF A$="П" THEN GOSUB 130
100 WEND
110 CLOSE 1
120 STOP
130 REM ПОДПРОГРАММА ПОИСКА (ЧТЕНИЯ) ЗАПИСИ
140 PRINT "ВВЕДИТЕ АРТИКУЛ ТОВАРА";
150 INPUT PC%
160 GET#1 ,PC%
275



170 PRINT "ЗАПИСЬ ТАКОВА: ";DES$;CVICSN$);CVS(P$)
180 RETURN
190 REM ПОДПРОГРАММА МОДИФИКАЦИИ ЗАПИСИ
200 GQSUB 130  :REM ЗАПРОС АРТИКУЛА И ИЗОБРАЖЕНИЕ ЗАПИСИ
210 PRINT "ДЛЯ ВНЕСЕНИЯ ИЗМЕНЕНИЙ В ОПИСАНИЕ ВВЕДИТЕ 1"
220 PRINT "                                3 КОЛИЧЕСТВО   -   2"
220 PRINT "                                В ЦЕНУ              -   3"
230 INPUT C*
240 ON C% GOTO 250,290,330
```

```
250 PRINT "ВВЕДИТЕ НОВОЕ ОПИСАНИЕ (ДО 20 СИМВОЛОВ): ";
260 INPUT D$
270 LSET DES$=D$
280 GOTO 360
290 PRINT "ВВЕДИТЕ НОВОЕ КОЛИЧЕСТВО ТОВАРА: ";
300 INPUT S%
310 LSET SN%=MKI$(S%)
320 GOTO 360
330 PRINT "ВВЕДИТЕ НОВУЮ ЦЕНУ;
340 INPUT P
350   LSET  P$ = MKS$(P)
360   PUT#1 , PC%
370 RETURN
380 END
```

8.6.3. ФАЙЛЫ ПРЯМОГО ДОСТУПА В БЕЙСИКЕ ВВС

Система ВВС поддерживает работу с файлами прямого доступа при наличии дисковода для гибких дисков. Полный идентификатор файла имеет следующий вид:

: дисковод.имя_справочника.имя_файла где:

дисковод - номер от 0 до 3, идентифицирующий физический дисковод;

имя—справочника — один символ, идентифицирующий отдельную группу файлов на диске;

имя_файла — имя файла, в состав которого входит до семи символов.

Ниже приведены примеры допустимых имен:

: 1.A.ONEPROG

: 0. $.DATA1

Если имя справочника и номер дисковода не указаны, то по умолчанию подразумеваются текущие значения этих параметров. Эти значения изображаются в распечатке со сведениями обо всех файлах на текущем диске, выдаваемой по команде *CAT. Эти значения могут быть изменены командами *DIR изменяет текущий справочник файлов *DRIVE изменяет текущий дисковод

Имена файлов в операторах OPENIN и OPENOUT относятся к текущему дисководу и текущему справочнику.

Оператор OPENIN открывает файл для чтения и записи в режиме прямого доступа, например:

```
10 N=OPENIN("DAFILE")
10   CHAN=OPENIN(A$)
```

где A$ содержит имя файла.

При создании файла оператором OPENOUT дисковая файловая система резервирует 64 сектора по 256 байт, т. е. 16К байт. Для каждого открытого файла в памяти ЭВМ резервируется буферная область размером в 256 байт. Одновременно могут быть открытыми до пяти файлов. Обмен данными между буфером и диском осуществляется автоматически в зависимости от значения, присвоенного PTR#.

При работе с файлами прямого доступа можно пользоваться всеми операторами, описанными в подразд. 8.4.4. и 8.4.7. В дополнение к ним предусмотрены новый файловый оператор и функция PTR#, позволяющая выбирать для последующего чтения или записи любой байт файла. Тем самым представляется возможность прямого доступа к области хранения файла на байтовом уровне. Например, оператор

100 PTR#M=10 установит указатель позиции в файле на байт 10, а оператор

100 PTR#M=NUM установит указатель позиции в файл на байт с номером NUM; оператор

100 PTR#M=PTR#M + 22 переместит указатель файла на 22 байта от предыдущего значения; оператор
100 PRINT PTR#M
напечатает текущее значение указателя файла. Обмены данными между буфером и диском осуществляются в зависимости от значения PTR#. Если оно равно 5000, то в буфер считывается 20-й сектор, содержащий байты с 4864-го по 5119-й, и чтение или запись будут происходить, начиная с байта 5000. Однако в пределах буфера указатель файла можно перемещать, не вызывая нового обмена с диском. Таким образом, концепция прямого доступа к файлу полностью ориентирована на побайтовую передачу данных. Значения вещественных числовых переменных записываются в пять байтов, а целых — в четыре, и перед каждой такой серией байтов записывается еще один байт, идентифицирующий тип значения. Строки записываются в виде последовательности составляющих их символов, перед которой записывается еще Два байта — один для идентификации строкового значения, а другой содержит длину строки. С помощью операторов BGET# и BPUT# значения можно считывать и записывать по одному байту. Оператор BPUT посылает один байт
файл в позицию, определяемую текущим значением указателя файла, а BGET читает байт из файла. Например, операторы
10 BPUT#N, 8

10 BPUT#N,A$
запишут значение 8 или первый символ строки A$ в позицию файла, определяемую текущим значением указателя. А оператор
10 X$=BGET#N считает байт из файла в X$.
Если в файле должна храниться, нерегулярная смесь значений числовых переменных, строк символов и байтов, то часть файла надо выделить под справочник, который будет считываться в массив тотчас же после открытия файла. В этом массиве должны регистрироваться начальные позиции всех элементов данных файла. В противном случае можно хранить формат логической записи с фиксированной длиной, содержащей несколько элементов. Например, логическая запись длиной в 19 байт может содержать два числовых значения по 5 байт в каждом и строку из 5 байт вместе с однобайтовыми заголовками у числовых значений и двухбайтовым заголовком перед строкой. Если номер такой логической записи (0, 1, 2, 3, ...) содержится в переменной REC, то для извлечения требуемой записи в буфер можно воспользоваться оператором
50 PTR#M=REC*19
В приводимом примере текущая позиция устанавливается на байт 100 файла DAFILE, а затем печатаются следующие 10 байт (символов) :
```
10 REM ПРИМЕР РАБОТЫ С ФАЙЛОМ DAFILE
20 N=OPENIN("DAFILE")
30 PTP#N=100
40 FOR I=1 TO 10
50  PRINT GET#N;
60  PTR#N=PTR#N+1
70 NEXT I
80 CLOSE#N
90 END
```
Операторы INPUT и PRINT действуют последовательным образом, но ими можно пользоваться для чтения и записи данных, начиная с позиции, на которую показывает текущее значение указателя файла, например:
50 PTR#N=50

```
60   INPUT#N, A, B, C$
```
или
```
50   PTR#N=1000
60   PRINT#N, A, B, C$
```
При аккуратной работе и, может быть, за счет сохранения в файле схемы размещения данных можно добиться достаточной легкости чтения и записи логически связанных наборов значений.
Файловая функция EXT#(N) возвращает длину файла в байтах:
```
L=EXT#(N)
```


8 6.4. ФАЙЛЫ ПРЯМОГО ДОСТУПА В. СИСТЕМЕ PET ФИРМЫ COMMODORE

Общий метод доступа к дисковым файлам в системе PET, а именно использование для канала дисковых команд вторичного адреса 15, считывание реакции на команды по этому же каналу, и использование для обмена данными каналов 2-14 уже были описаны в подразд. 8.4.6.
При работе с файлами прямого доступа требуется явным образом пользоваться буфером в 256 байт, выделяемым для каждого файла. Для этого с помощью операторов PRINT по каналу команд 15 надо потребовать от дискового контроллера передачи блока с диска в буфер или обратно. Данные могут быть считаны в буфер или записаны из него в файл с помощью операторов INPUT# и PRINT#.
Способ доступа к диску посредством команд довольно сложен. Он выполняется на основе адресации по дорожкам и секторам, причем каждый сектор содержит 255 байт данных и на 35 дорожках стандартного дисковода (модели 3040) помещается от 17 до 21 секторов. Таким образом, если Вы храните по одной записи в блоке размером 255 байт, то Вам обычно надо ссылаться на конкретную запись по ее номеру, который программа должна преобразовывать в номер дорожки и номер сектора. Максимальное число доступных блоков равно 670. По желанию в одном блоке можно хранить по две записи или более. Так как в буфере предусмотрен указатель, обеспечивающий чтение или запись отдельных частей буфера, начиная с отмеченной указателем позиции, то с его помощью можно получать доступ к отдельным записям из блока. Информация располагается на диске по направлению от внешней дорожки с номером 1 к внутренней дорожке с номером 35 следующим образом: каждая из дорожек с 1-й по 17-ю содержит по 21 сектору; дорожка 18 содержит 20 секторов, но пользоваться ею нельзя, так как на ней хранится справочник файлов; дорожки с 19-й по 24-ю содержат по 20 секторов, дорожки с 25-й по 30-ю — по 18 секторов, а дорожки с 31-й по 35-ю — по 17 секторов. Всего получается 690 секторов, из которых для данных, за вычетом 20 секторов справочника файлов, остается 670 секторов.
В дисковой системе PET предусмотрен богатый набор команд, позволяющих выполнять очень сложные манипуляции. Ниже приведена схема работы с диском в предположении, что в каждом блоке хранится по одной записи. При этом отдельные действия могут быть реализованы в виде подпрограмм Для применения в больших программах.
(а) Открыть файл прямого доступа с логическим номером С и для большего удобства со вторичным адресом С в диапазоне 2 ... 14:
```
10  OPENC,8,C"#"              (открыть файл и зарезервировать буфер)
20  OPEN 15, 8, 15             (открыть канал команд)
```
(проверить реакцию системы на наличие ошибки)
(б) Считать запись. Вначале номер записи преобразуется в номер дорожки и номер сектора S; номер дисковода D должен быть равен 0 или 1:

(преобразовать номер записи в T и S)
100 PRINT#15,"U1:"C;D;T;S (считать блок с диска в буфер)
110 PRINT#15,"B-P:"C; 1 (установить указатель буфера на его начало)
(проверить реакцию системы на наличие ошибки) 120 INPUT#C, список элементов (прочитать значение из буфера)
(в) Сделать запись: (преобразовать номер записи в T и S)
200 PRINT#15,"U1:"C;D;T;S (считать блок с диска в буфер)
210 PRINT#15,"B-P:"C;1 (установить указатель буфера на его начало)
(проверить реакцию системы на наличие ошибки) 220 PRINT#C, список элементов (записать данные в буфер)
230 PRINT#15,"U2:"C; D; T; S (вывести буфер на диск)
(проверить реакцию системы на наличие ошибки)
(г) Закрыть файл:
300 CLOSE C 310 CLOSE 15

8.7. ЭКСПЛУАТАЦИЯ ФАЙЛОВ

Методы обработки данных в больших файлах гораздо сложнее простых приемов, описанных ранее в данной главе. Поэтому в настоящем разделе предлагается несколько советов тем, кто хочет разрабатывать на Бейсике программы для обработки данных.

Существенной частью эксплуатации файлов являются поиск отдельной записи и модификация записей. Модификация состоит в добавлении новых документов или в удалении либо изменении уже существующих.

Термин "обработка данных" обычно связывают с наличием большого объема данных, хотя слово "большой", вообще говоря, не поддается количественной оценке. Нетрудно видеть, что как только объем данных вырастает до такой степени, что они уже не помещаются в памяти ЭВМ, их объем можно рассматривать как большой; при этом для сортировки и поиска потребуются совершенно иные методы по сравнению с теми, которые уже обсуждались в предыдущих главах.

Содержащиеся в файлах прямого доступа данные концептуально аналогичны данным, хранящимся в массивах, и поэтому к таким файлам достаточно хорошо применимы описанные ранее методы работы с массивами. Если Вам требуется очень эффективный механизм поиска в файлах прямого доступа, сводящий число обменов с диском до минимума, то можно применить хэшированную адресацию записей в файле, согласно которой ключ записи (поле записи, однозначно ее идентифицирующее) преобразуется в число называемое хэшированным адресом, которое используется как номер записи для хранения ее в файле. Этот метод хорош до тех пор, пока два разных клю-

ча записи не приведут к одному и тому же числу; в этом случае для помещения новой записи в файл применяют иной метод. Один из способов состоит в помещении новой информации в свободную запись, ближайшую к уже занятой Конечно, механизм извлечения данных обязан учитывать эту возможность при поиске требуемой записи.

Простой метод хэширования для поля, содержавшего фамилию, состоит в сложении кодов ASCII всех букв и делении полученной суммы на простое число, несколько превышающее максимально возможное число записей. Остаток от деления представляет собой число, лежащее между 0 и этим простым числом; его-то и можно взять в качестве хэшированного адреса данной записи. За детальными сведениями можно обратиться к многочисленной литературе.

Для хранения большого числа данных наиболее широко используются последовательные файлы, чаще всего либо из стоимостных соображений, либо по той причине, что размеры файлов прямого доступа могут быть несколько ограничены по системным причинам или из-за принятой технологии работы с ними.

Быстрого способа локализации записей в последовательном файле нет, и при поиске требуемой записи файл приходится просматривать последовательно. Однако можно уменьшить среднее время поиска, если накопить запросы на поиск и на один просмотр файла выбрать все записи, удовлетворяющие условиям поиска целого списка запросов. Если, скажем, за один полный просмотр файла можно удовлетворить десять запросов, то в среднем такой процесс поиска окажется гораздо быстрее индивидуального выполнения каждого из десяти запросов. Возможность применения этого метода зависит от того, можно ли подождать с ответом на запрос до тех пор, пока не будет накоплен и обработан подобный список.

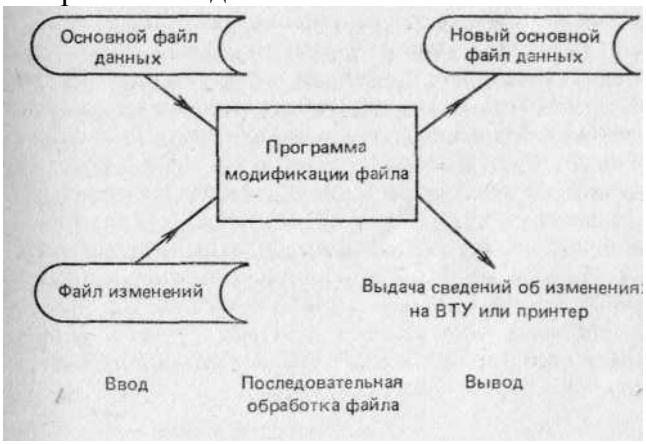


Рис. 8.9. Модификация основного файла. Если модификация прошла удовлетворительно, то исходный файл заменяется на новый основной

281

По описанной выше схеме осуществляется и модификация последовательного файла. Часто изменения основного файла происходят не при каждом запросе на изменение элементов данных: запросы накапливаются в списке и вносятся в основной файл данных за один прием (рис. 8.9). Список изменений обычно помещается в память ЭВМ, но его можно хранить и в специальном файле, "файле изменений".

Обычно требуется, чтобы записи основного файла были отсортированы по значениям некоторых ключевых полей, поэтому наиболее эффективный метод внесения изменений предполагает сортировку изменяемых элементов данных. Внесение списка изменений в основной файл во многих отношениях подобно слиянию двух файлов. Если оба файла отсортированы, то и результирующий новый основной файл также будет отсортирован. Существует следующий путь построения большого отсортированного файла из неупорядоченных данных: надо многократно вносить отсортированные списки изменений, начиная с пустого основного файла.

Один из основных методов сортировки последовательного файла использует слияние нескольких (более чем двух) файлов, каждый из которых со- I держит много групп записей. Каждая группа первоначально способна поместиться в памяти ЭВМ, где ее можно отсортировать внутренним образом. Этот метод называется *многофазным слиянием;* более детально о нем можно узнать в книге Peter Naur *Concise Survey of Computer Methods,* выпущенной издательством Studentliteratur (Лунд, Швеция) в 1974 году.

Нередко достаточно хорошими оказываются и некоторые частные методы. Например, если в распоряжении имеются только два последовательных файла, каждый на своем кассетном магнитофоне, то для сортировки данных в большом файле пригоден следующий метод. Часть данных

считывается из файла 1 в память ЭВМ и сортируется там, а затем выводится в файл 2. Этот процесс повторяется; когда будет достигнут конец файла 1, то в файле 2] будут находиться небольшие группы отсортированных (внутри группы) данных. Повторите этот процесс, переписав данные из файла 2 в файл 1, но использовав при этом другой размер рабочей области памяти, предназначенной для внутренней сортировки. Повторите еще раз, переписывая данные из файла 1 в файл 2 и т. д. Пусть при переносе данных в одном направлении, скажем из файла 1 в файл 2, область памяти для внутренней сортировки вмещает одно и то же число записей, например 100, а при переносе в обратном **направлении** число сортируемых в памяти записей каждый раз меняется (скажем, чередуется от 51 до 79). В противном случае возникает периодичность и полной сортировки файла данных никогда не произойдет; в результате он окажется разбитым на отсортированные отрезки, длина которых равна произведению размеров двух областей внутренней сортировки. Продолжайте процесс переноса данных с одной ленты на другую до тех пор, пока все данные не будут отсортированы. Это очень длительный процесс, но в конце концов он приводит к требуемому результату.

ЧАСТЬ IV. ОСНОВНЫЕ ИТОГИ

Эта часть рассчитана на тех, кто закончил чтение части I. В ней приводятся полезные сведения о работе с ЭВМ и Бейсиком.

9. РАЗРАБОТКА ПРОГРАММ

Наряду с новым материалом в этой главе собрано воедино несколько тем, обсуждавшихся ранее для иллюстрации непосредственных и очень полезных возможностей разработки программ, обеспечиваемых Бейсиком. Как и в других главах, предостерегаем, что не все версии Бейсика обладают всеми описанными ниже возможностями.

Процесс разработки программы рано или поздно доходит до этапа отладки. Совершаемые в процессе отладки действия с большим трудом поддаются описанию. Мастерство отладки оттачивается в основном на практике, и у некоторых программистов, по-видимому, развивается интуиция выявления ошибок в программе, передать которую другим невозможно. Первые два раздела настоящей главы посвящены некоторым простым методам выявления ошибок, которые могут помочь Вам начать разработку программ, а в третьем разделе описан один из способов разработки сложных программ.

9.1. ОТЛАДКА

Отладкой называется выявление и устранение ошибок в программе.

9.1.1. ПРОСТЫЕ ОШИБКИ

Наиболее часто встречаются ошибки, порожденные простыми опечатками при наборе текста программы или некоторым недопониманием действия некоторых операторов Бейсика, например циклов FOR-NEXT. Некоторые опечатки обнаруживаются системой при проверке синтаксиса каждого оператора. Другими словами, система тщательно проверяет, является ли каждый оператор программы допустимым оператором Бейсика. В некоторых системах с Бейсиком (наиболее удобных для начинающих) проверка осуществляется тотчас же после ввода строки, например:

```
10  Y=6*X+5Z+2        (нажмите клавишу возврата каретки или SEND)
SYNTAX ERROR
```

В этом случае строку надо исправить на

```
10  Y=6*X+5*Z+2
```

Но если вместо этой строки будет набрано

```
10  Y=X*6+Z5+2
```


то сообщение об ошибке не появится, так как строка синтаксически правильна, хотя на самом деле должна быть

```
10  Y=X*6+Z*5 + 2
```

Многие системы с Бейсиком проверяют наличие синтаксических ошибок только после запуска программы по команде RUN. Это не слишком удобно для начинающих программистов, которые чаще всего делают тривиальные ошибки и в результате получают множество сообщений об ошибках. Этого бывает достаточно для того, чтобы будущий программист потерял надежду добиться успеха. Если Вы сами окажетесь в подобной ситуации, то придерживайтесь золотого правила сконцентрироваться на одном или двух первых сообщениях в списке ошибок и совершенно игнорировать все остальные. Часто бывает так, что из-за неправильного оператора в начале программы последующие операторы могут быть сочтены системой ошибочными, хотя по сути они правильны. Устранение одной-двух первых ошибок может привести к исчезновению многих других; поэтому при новом запуске программы по команде RUN могут остаться сообщения только об истинных ошибках.

Один из видов ошибок, вызывающих такой "сногсшибательный" эффект, может заключаться в неправильном определении переменных в начале программы. Например, определение

10 DIM AA(10),B1 (20,5), C1 (2,3) при наличии в программе операторов вида 100 BB(16,1)=2* C(1)+20
может явиться причиной недоразумений из-за того, что в качестве массива был определен B1 (,), а не BB и C1 (,), а не C ( ). Либо Вы ошиблись в написании имени массива при его определении в строке 10, либо собирались использовать новые массивы в строке 100. Из-за подобной небрежности легко угодить в ловушку, так как, по мнению автора, Бейсик слишком "великодушен" при обнаружении не определенного ранее массива: вместо сообщения об ошибке он предполагает, что по умолчанию одномерный массив имеет размер 10, а двумерный - 10,10. Если Вы не собирались использовать новый массив и набрали BB по ошибке вместо B1, то сообщения об ошибке не получите.
Подытожим: синтаксическая проверка обнаруживает обычные ошибки и опечатки в любых служебных словах, например INPUT, PRINT и т. д., но, возможно, осуществляется только после запуска программы по команде RUN. Поэтому нередко рекомендуется попробовать любую конструкцию, в которой Вы не уверены, в программе всего из нескольких строк, возможно, никак не связанной с программой, которую требуется разработать. Чаще всего трудности связаны с циклами FOR-NEXT и массивами.
В цикле FOR-NEXT Вы не должны изменять управляющую переменную — система заботится об этом сама. Если возникают какие-то сомнения, всегда

включайте операторы PRINT для демонстрации происходящих в программе действий:

```
10 REM ТЕСТ ЦИКЛА
20 INPUT "НАЧАЛЬНОЕ ЗНАЧЕНИЕ";S
30 INPUT "КОНЕЧНОЕ ЗНАЧЕНИЕ";T
40 INPUT "ШАГ";S1
50 PRINT "ЦИКЛ FOR I=";S;"TO";T;"STEP";S1
60 FOR I=S TO T STEP S1
70   PRINT "В ЦИКЛЕ FOR I=";I
80 NEXT I
90 PRINT "ВЫХОД ИЗ ЦИКЛА"
100 END
```

Эта программа несколько замысловата, но в точности демонстрирует то, что происходит в цикле FOR-NEXT. Даже простого текста, например

```
10   FOR 1 = 1 TO 5 20     PRINT I
30  NEXT I
```

может оказаться достаточным для устранения всех неясностей при изучении циклов. Придумайте аналогичные простые программы, демонстрирующие то, что происходит при использовании элементов массивов. Вводите значения, изменяйте элементы массива и распечатывайте все изменения, происходящие на каждой стадии.
Если одно обстоятельство, которое надо учитывать при поиске ошибок, состоит в том, что Бейсик выдает очень сжатые сообщения о характере ошибки. Это связано с тем, что правильная диагностика ошибки может быть довольно сложной, для чего требуется больший компилятор/интерпретатор языка Бейсик. Предусмотреть хорошую диагностику ошибок вполне возможно и крайне желательно, но очень немногие компиляторы/интерпретаторы любого языка программирования оказываются удовлетворительными в этом отношении. Поэтому новому пользователю любой системы с Бейсиком приходится самому методом проб и ошибок выяснять, что же могут означать сообщения об ошибках. Это является еще одним доводом в пользу составления пробных программ из нескольких операторов вместо того, чтобы написать длинную программу и только потом начинать ее отлаживать.

Наконец, не забывайте проверять такие простые опечатки, как замена 1 на I или 1 (букву) или О (буквы) на 0 (цифру). Оки часто происходят при наборе этих символов в комбинациях с другими символами, что затрудняет обнаружение разницы; например, АО может быть написано вместо А0 или I=1+1 вместо I=I+1.

9.1.2. ДИАГНОСТИКА

После того как система с Бейсиком сочла программу правильной и запуск программы привел к каким-либо результатам, ее надо проверить, использовав такие входные данные, для которых результаты известны заранее. Это основное условие приемлемости программы. Тестовые данные должны обеспечивать проверку всех или как можно большего числа различных путей в про-

грамме, по которым может пойти обработка данных. Модульный подход к программированию позволяет достаточно легко проверять отдельные части программы, однако при этом остерегайтесь пользоваться в модулях глобальными переменными. Неумышленные изменения таких переменных в одном из модулей могут вызвать ошибки в работе других, внешне правильных модулей.

Состояния переменных программы и пути ее выполнения всегда можно изобразить с помощью операторов PRINT, которые так легко добавить в программу на Бейсике и удалить из нее. Не забывайте о возможности немедленного исполнения операторов PRINT (набранных без номера строки, см. разд. 2.4), позволяющих изображать значения переменных после прекращения работы программы из-за ошибки или в результате выполнения оператора END. Например, пусть программа вычисления корней квадратного уравнения выдает ошибочные результаты. Квадратное уравнение вида

$ax^2 + bx+c=0$ имеет два корня:

$$x_1 = \frac{-b+\sqrt{(b^2-4ac)}}{2a}$$

$$x_2 = \frac{-b-\sqrt{(b^2-4ac)}}{2a}$$

Ошибочная программа такова:

```
10 REM РЕШЕНИЕ КВАДРАТНОГО УРАВНЕНИЯ 20 INPUT "КОЭФФИЦИЕНТЫ";A,B,C
30 D=B*B-4*A*C
4в IF D<0 THEN 100
50   D=SQR(D)
6Э   R1=-B+D/2*A
70   R2=-B-D/2»A
80   PRINT "ДВА КОРНЯ УРАВНЕНИЯ РАВНЫ";R1;"И";R2
90 STOP
100 PRINT "ИМЕЮТСЯ ТОЛЬКО МНИМЫЕ КОРНИ" 110 END
RUN
КОЭФФИЦИЕНТЫ ?1.-5.6
ДВА  КОРНЯ  УРАВНЕНИЯ  РАВНЫ  5.5в  И  4.5в
```

Эти результаты ошибочны, так как корни данного квадратного уравнения x^2 -5x + 6=0 равны 2 и 3 (нетрудно проверить, что (x-2) *(x* -3) =x^2 -5x+6). Попробуйте распечатать внутренние значения с помощью немедленно исполняемого оператора PRINT:

```
PRINT D; A; B; C          1   1 -5   6
```

Все они правильны; поэтому теперь надо проверить правильность вычислений R1 и R2 в строках 60 и 70. В данном случае нельзя распечатать никакой

другой полезной информации, но в более сложных программах может понабиться проверка значений большого числа переменных. Попробуйте проделать вычисления, приводящие к значению R1:
R1=-(-5)+1/2*1=5+1/2, в то время как правильным было бы

$$R1 = \frac{-(-5)+1}{2*1} = \frac{5+1}{2} = 3.$$

Таким образом, строки 60 и 70 должны иметь вид
60 R1 = (-B+D)/(2*A) 70 R2=(-B-D)/(2*A)

9.2. ВЫЯВЛЕНИЕ ОШИБОК

В предположении, что синтаксические ошибки рано или поздно будут локализованы с помощью информации, предоставляемой компилятором/интерпретатором, настоящий раздел посвящен средствам локализации ошибок, возникающих во время исполнения программы. Во многих системах такие ошибки приводят к прекращению исполнения программы и изображения сообщения об ошибке или номера ошибки. Так как ничего другого не остается, то для выявления причины ошибки программист должен полагаться на операторы PRINT и тщательную проверку текста программы. Если системой выдается только номер ошибки, то с помощью набора специального символа или особой команды можно получить полное сообщение об ошибке. Например, в системе BBC такой командой служит REPORT.

В следующих разделах описаны два других основных средства, очень полезных для выявления ошибок. Первое из них обеспечивает отслеживание (трассировку) номеров выполняемых операторов программы. Второе позволяет программе самой реагировать на возникновение ошибки и выполнять по этому поводу определенные действия. Это довольно экзотическая возможность; она рассчитана не на обработку "обычных" ошибок, возникающих при разработке программ, а на устранение ошибочных ситуаций при эксплуатации программы. Такие ситуации могут быть вызваны ошибками в наборе данных на клавиатуре, электрическими помехами в соединительных кабелях, в результате чего программе могут передаваться "странные" символы. И в этом случае вместо останова и перезапуска программы можно выполнить определенные действия и продолжить ее исполнение.

9.2.1. ТРАССИРОВКА ИСПОЛНЕНИЯ ПРОГРАММЫ

В программах, где довольно много операторов IF, циклов FOR-NEXT и подпрограмм, образуется значительное число путей, по которым может пойти их исполнение. Если не выводить промежуточные результаты с помощью операторов PRINT, то при возникновении ошибки трудно определить, по какому именно пути шло перед этим исполнение программы. Некоторые системы обеспечивают удобный способ получения подобной информации с

помощью команды TRACE (трассировка), побуждающей интерпретатор печатать номер каждого выполняемого оператора. Тем самым при получении сообщения об ошибке складывается полная "предыстория" исполнения программы.

```
10 REM ДЕМОНСТРАЦИЯ ТРАССИРОВКИ
20 TRACE
30 I=10
40 FOR J=l TO 2
50   I=I+1
60   PRINT J;1
70 NEXT J
```

```
80 END
RUN
[30]  [40]  [50]  [60]  1  11
[70]  [50]  [60]  2  12
[70]  [80]
OK
```

Приведенная выше программа демонстрирует действие оператора TRACE в Бейсике Microsoft, которым можно пользоваться и как командой, в режиме немедленного исполнения. Номера операторов изображаются заключенными в квадратные скобки с тем, чтобы их легко было отличить от других данных, которые могут выводиться в диагностических целях, например от значений J и I, выдаваемых в строке 60. Оператор NO TRACE отменяет трассировку.

В Бейсике BBC для управления трассировкой используются операторы -TRACE ON (включить трассировку) и TRACE OFF (выключить трассировку) и, кроме того, предусмотрен оператор TRACE N, где N - номер оператора. В этом случае проводится трассировка только тех операторов, номера которых не превышают N.

При трассировке значительно замедляется исполнение программы и может создаваться множество выходных данных. Чтобы избежать этого, можно сначала подождать, пока программа не окажется вблизи ошибочной области, приостановить ее исполнение (клавишей "прерывание"), затем набрать команду TRACE и продолжить работу программы по команде CONTINUE. Другой способ состоит в окаймлении ошибочной области в программе операторами TRACE и NO TRACE.

Учтите, что интерпретатор может оптимизировать предоставляемую ему для исполнения программу на Бейсике, поэтому могут изображаться не все ожидаемые номера строк. Это относится к оператору FOR, открывающему цикл FOR-NEXT в приведенном выше примере.

9.2.2. ПРОЦЕДУРЫ ОБРАБОТКИ ОШИБОК

Эту тему повышенной трудности при первом чтении можно пропустить, пока не будет прочитана часть III.

Как уже говорилось во введении к данному разделу, процедуры обработки ошибок -довольно экзотическая возможность, рассчитанная на то, чтобы позволить "правильной" программе самой обрабатывать ошибки, которые могут появиться в окружающей ее программной и аппаратной среде из множества источников.

МикроЭВМ, будучи небольшими и транспортабельными, зависят от местных сетей электропитания. В отличие от больших ЭВМ, которые могут иметь стабилизированное питание микроЭВМ должны подсоединяться к обычной электросети, в которой вполне могут появляться "выбросы" напряжения из-за работы других электроприборов, а также флуктуации уровня напряжения. Кроме того, под воздействием радиочастотного электромагнитного излучения в неэкранированных кабелях или коннекторах могут возникать электрические шумы.

Транспортабельность - новое качество ЭВМ, которое приводит к тому, что все больше и больше неспециалистов начинают использовать микроЭВМ. Это хорошее дело, но в результате вероятность ошибочного ввода значительно увеличивается. Хотя при работе в таких условиях вводимые значения должны тщательно проверяться, тем не менее в отдельных случаях ошибочное значение может попасть в программу и через некоторое время привести к серьезной ошибке.

Только опыт позволит определить ошибки, которые могут возникать по двум описанным выше причинам, и найти способы устранения их влияния. Ниже описан механизм процедур обработки ошибок, а разработка подходящей процедуры должна ориентироваться на конкретную ситуацию.

В нормальном состоянии программы ее исполнение прекращается при возникновении ошибки. При этом значения внутренних переменных обычно сохраняются и их можно проанализировать с помощью немедленно исполняемого оператора PRINT. Однако можно определить глобальное условие, согласно которому при возникновении ошибки происходит передача управления специальной процедуре обработки ошибок. Это условие имеет вид

```
10   ON ERROR GOTO 500
```

где строка 500 служит началом раздела обработки ошибок. Пока действие этого оператора остается в силе, при возникновении ошибки управление передается процедуре обработки ошибок, которая в конце концов либо возвращает управление в то место, где возникла ошибка, либо в какое-либо иное место программы. Оператор ON ERROR обладает "отложенным" действием, что означает, что он вступает в дело тогда, когда в каком-либо месте программы произошла ошибка. Для перехода к нормальному режиму исполнения вылавливание ошибок может быть прекращено оператором ON ERROR OFF или ему подобным. (В Бейсике Microsoft для этих целей используется оператор ON ERROR GOTO 0.)

Следующий пример иллюстрирует процедуру вылавливания ошибок:

```
10 REM
20    . . .
30  ...
40   ON  ERROR  GOTO   200
50 . . .
200 REM ПОДПРОГРАММА ОБРАБОТКИ ОШИБКИ
210 IF ERR=11 THEN 240
220 PRINT "ОШИБКА";ERR;"В СТРОКЕ";ERL
230 STOP
240   PRINT "ЗАФИКСИРОВАНА ПОПЫТКА ДЕЛЕНИЯ НА НУЛЬ. ";
245   PRINT "ПРОВЕРЬТЕ ВВОД"
250   Z=1.0E-20
260  RESUME
289
```

В Бейсике Microsoft и в Бейсике BBC имеются две переменные (или функции), ERR и ERL. Первая содержит номер ошибки, а вторая - номер строки. В приведенном выше примере проверка ошибок была инициирована в строке 40, после чего при появлении ошибки управление было передано строке 200. Этой строкой начинается специализированная подпрограмма обработки ошибки, возникающей при попытке деления на 0 поэтому если произошло что-то иное, то информация об этом печатается в строке 220. Если обнаружено деление на 0, то в строке 240 выводится соответствующее сообщение и значение переменной (Z) устанавливается таким образом, чтобы ошибки не возникло Оператор в строке 260 вызывает возврат управления к ошибочной строке и повторение ее выполнения.

Чтобы в процедуре обработки ошибок можно было выяснить детали ошибки, в системе должны быть предусмотрены переменные типа ERL, ERR или специальные операторы IF. Как только выполнены соответствующие действия по обработке ошибки, которые могут включать в себя и прекращение вылавливания будущих ошибок, управление исполнением возвращается обратно в программу либо с помощью оператора RESUME, либо, если такого нет, с помощью оператора GOTO. Оператор RETURN использовать нельзя, так как процедура обработки ошибок вызывается не таким способом, как обычные подпрограммы.

9.3. СТРАТЕГИЯ РАЗРАБОТКИ

Одной из целей данной книги, подобно многим другим книгам о языках программирования, было описание свойств языка и действия его операторов. Наряду с этим в ней была сделана попытка научить методам разработки и составления программ.

Основной трудностью разработки является неясность деталей решения задач; поэтому пропагандируемый нами метод уточнения принадлежит к числу тех, при использовании которых надо стараться как можно дольше откладывать принятие решений, зависящих от этих деталей. Однако даже довольно общие решения, принятые в самом начале, имеют тенденцию включать в себя определенные черты последующих уровней детализации. Поэтому здесь не такая большая свобода выбора, как можно было себе предоставить по описанию метода разработки. Действительно, различные варианты решения одной и той же задачи, рассматриваемые на верхних уровнях детализации, скорее всего, не приведут к одному и тому же решению. Только опыт может показать, какой из подходов похож на "лучшее" решение, если такое существует.

Автор пришел к следующему эмпирическому методу, который позволяет составить определенное мнение о деталях решения и может быть полезен в различных ситуациях.

Решаемая задача может включать в себя некоторые незнакомые методы обработки данных, например манипулирование файлами, или организацию сложных вложенных циклов, или выдачу на печать в специальном формате. В этом случае лучше всего освоить новые методы, исполнив простые программы, и вначале разработать прототип конечного продукта. Начните, как обычно, с наброска программы, который может включать в себя только ту часть решения, которая представляется неясной, а затем введите его в ЭВМ и попробуйте исполнить. Отладьте программу и не жалейте времени на ее

модификацию до тех пор, пока не убедитесь в том, что получено наилучшее решение. На этом этапе не пытайтесь добавить к только что разработанной

части программы оставшуюся часть решения. Если Вы сделаете такую попытку. то обнаружите, что структура полученной программы не является

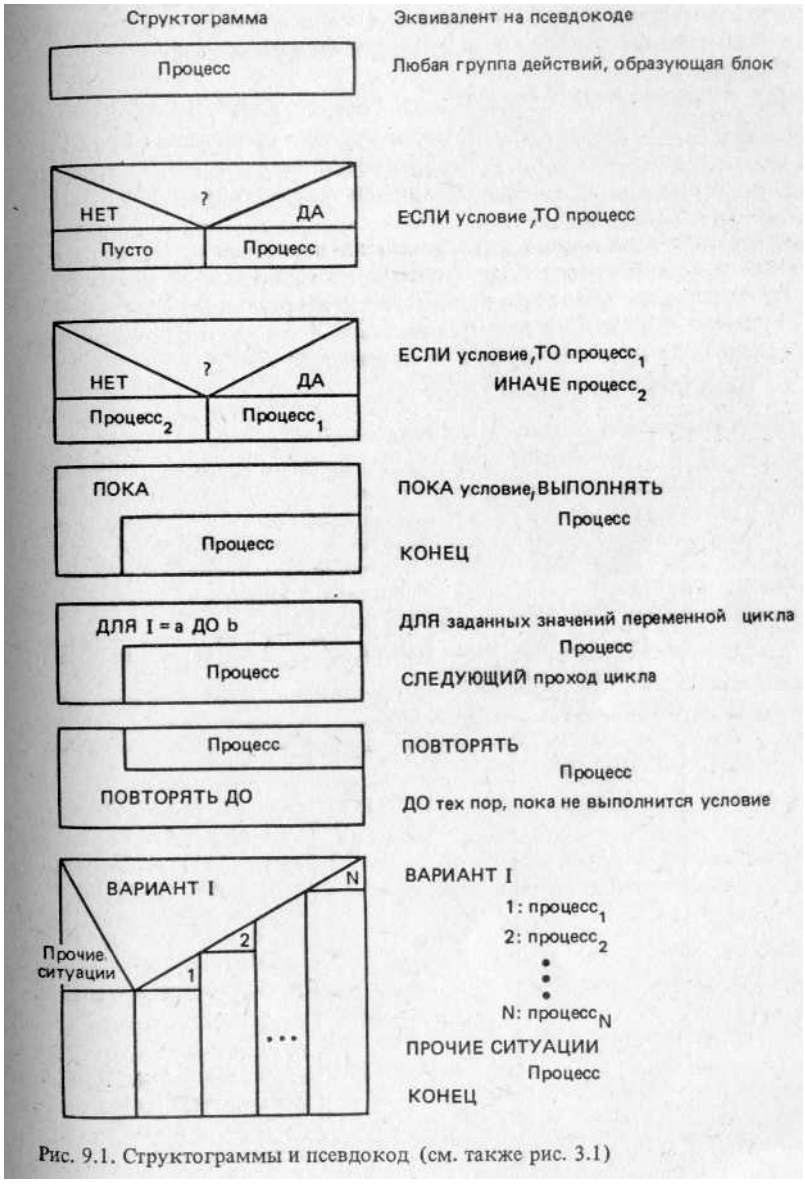


Рис. 9.1. Структограммы и псевдокод (см. также рис. 3.1)

сколько-нибудь элегантной и понятной, а напоминав- беспорядочную "'кучу" Оставьте на какое-то время работу над решением, скажем на ночь, с тем чтобы Вы могли отвлечься от деталей запрограммированной части решения и начать на свежую голову разработку полного решения. На этом этапе согласуйте все надлежащие детали проверки ошибок, взаимодействия программы с пользователем и т. д. В результате должно получиться кое-что достойное гордости и заслуживающее передачи друзьям и коллегам. Подытожим:

(а) Примените метод разработки "сверху вниз", использовав структограммы или псевдокод. Основные конструкции приведены на рис. 9.1.

(б) Поэкспериментируйте, выбирая несколько отправных точек для разработки и проработав все трудные детали так, как описано выше. Попробуйте выполнять их на ЭВМ.

(в) После выполнения экспериментов вернитесь к (а) и разработайте полный и четкий проект программы. Воплотите его в программу для ЭВМ.

(г) Соберите и приведите в порядок все использованные при работе бумага, включив записи всех потребовавшихся формул, и сохраните их, приложив распечатку программы, если ее можно получить.

9.4. МАШИННЫЙ КОД И КОМПИЛЯТОРЫ

(При первом чтении этот раздел лучше пропустить)

Команды, которые способна распознать и выполнить аппаратура ЭВМ, называются *машинными кодами.* Они соответствуют элементарным действиям, на выполнение которых рассчитана машина, и представляются ее центральному процессору как двоичные данные, состоящие из одного или нескольких байтов. Каждый производитель разрабатывает собственную модель ЭВМ, и в результате многие модели имеют свои уникальные машинные коды. В левом столбце табл. 9.1 приведено несколько примеров машинных кодов для микропроцессора 8080 фирмы Intel.

Таблица 9.1. *Примеры машинных кодов и кодов на языке ассемблера для микропроцессора Intel 8080*

Машинный код в двоичном виде		Код ассемблера	Интерпретация
Первый байт 0011 1110	Второй байт 1111 1111	MVI A,11111111 B	Очищает аккумулятор (называемый А) и заносит в него двоичное значение 1111 1111, на что в языке ассемблера указывает буква В. Аккумулятор содержит восемь битов и служит для выполнения арифметических и логических операций. Команда занимает два байта
1000 0110		ADD M	Добавляет содержимое ячейки памяти (восемь битов) к значению, находящемуся в аккумуляторе. Адрес ячейки памяти находится в регистре по имени М. Команда занимает один байт

Таблица 9.1 (окончание)

Машинный код в двоичном виде	Код ассемблера	Интерпретация

Первый байт Второй байт 1101 0011 0000 0001]	OUT 00000001 B	Копирует находящееся в аккумуляторе значение в определенные выводящие проводники, называемые шиной данных. Число 0000 0001 определяет требуемое устройство вывода и выдается на набор проводников, называемый шиной адреса. После этого за выполнение вывода данных принимаются другие компоненты ЭВМ. Команда занимает два байта

Как нетрудно видеть, машинный код очень трудно записать правильно, потому что при работе с двоичными числами легко сделать ошибку. На самом деле для набора на клавиатуре двоичные числа можно записывать в гораздо более компактной форме — в виде шестнадцатеричных чисел, но это не снимает всех проблем, связанных с программированием в машинных кодах. Чтобы исключить некоторые из этих проблем, придумано более простое представление команд, в котором двоичные коды заменены мнемоническими, более легкими для чтения и запоминания. Последние вместе с несколькими другими полезными средствами программирования образуют то, что называется *языком ассемблера.* Центральный столбец табл. 9.1 иллюстрирует некоторые команды языка ассемблера. Благодаря тесной связи между языком ассемблера и машинными кодами нетрудно разработать программу, воспринимающую текст на языке ассемблера в качестве входных данных и выдающую машинные коды. Такая программа называется ассемблером. Она работает следующим образом: во-первых, должна быть написана программа, состоящая из команд на языке ассемблера, которые вводятся в ЭВМ; во-вторых, ассемблер проверяет вводимые данные и выполняет трансляцию (перевод), дающую на выходе машинные коды. Наконец, эти машинные коды загружаются в подходящую часть памяти ЭВМ и исполняются.

Хотя на языке ассемблера написано много программ, программирование на нем имеет ряд недостатков: программы получаются довольно длинными, их разработка требует известного мастерства и тесно связана с особенностями функционирования аппаратной части конкретной ЭВМ. Например, для выполнения действий, описываемых Фрагментом программы на Бейсике

```
10  INPUT A,B
20  LETC=A*B
30  PRINT C
```

а именно для перемножения двух чисел, потребуется по меньшей мере 100 команд на языке ассемблера для микропроцессора 8080 фирмы Intel.

Хотя при трансляции в машинные коды программы, написанной на языке высокого Уровня типа Бейсика, действия компилятора аналогичны действиям ассемблера, тем не менее дистанция между входными языками настолько велика, что компилятор оказывается гораздо более сложной программой.

Существует много вариантов написания компилятора. Сущностью диалогового языка, подобного Бейсику, является способность легкого редактирования программ, распечатывания программ по мере необходимости и исполнения программ по команде Все эти операции могут выполняться в любом порядке; обычно они многократно повторяются при разработке и модификации программы.

Один из подходов, называемый чистой компиляцией, состоит в преобразовании машинные коды каждой строки программы на Бейсике по мере ее получения. По окончанию ввода программы результирующие машинные коды полностью готовы к исполнению. Однако для выполнения такой построчной трансляции требуются большие таблицы и списки, в которых хранятся сведения об исходных именах переменных и номерах операторов, позволяющие компилятору гарантировать выдачу полной программы. Например, вплоть до набора строки 510 должна храниться информация о том что ранее была введена строка 10 GO TO 510. Предыдущие строки соответственно могут изменяться или удаляться вплоть до завершения программы. Этот подход обеспечивает создание машинных кодов, готовых к исполнению, но из-за требования, чтобы процесс компиляции проходил в режиме диалога, приходится занимать в памяти много места для хранения очень больших таблиц.

Другой подход, называемый чистой интерпретацией, состоит в хранении программы на Бейсике в том виде, в котором она была введена в ЭВМ, и в построчной трансляции программы в момент ее исполнения. Таким образом, если встретился цикл, то его операторы будут повторно транслироваться при каждом проходе цикла, так как система транслирует каждый очередной оператор и не сохраняет полученные машинные коды (которые, скорее всего, и не образуются). В этом случае за удобство редактирования и распечатывания введенной программы приходится расплачиваться очень малой скоростью исполнения. В то время как компиляторы могут проверить каждый оператор по мере его ввода и немедленно сообщить об ошибке, интерпретаторы могут делать это только во время исполнения программы, что является серьезным недостатком.

Поставляемые системы трансляции находятся где-то между двумя описанными выше системами. На рис. 9.2 показаны основные действия подобной системы, которая в литературе может называться компилятором или интерпретатором. В ней используются некоторые типы внутреннего представления программы, обеспечивающего построчную обработку программы на Бейсике, но содержащего много приготовлений, необходимых для последующего исполнения программы. Ниже показано несколько строк программы на Бейсике вместе с ее внутренним форматом, используемым в ЭВМ ICL 2904. Исходная программа имеет вид

```
10   REM ПРОСТОЙ ПРИМЕР
20   P =3.14159
30   INPUT R
40   A=P*R*R
50   C=2.0*P*R
60   PRINT R,A,C
70   END
```

а ее внутренняя форма такова:

```
K! ПРОСТОЙ ПРИМЕР
P'3.14159K =
L!R'
A'P'R'*R'*K=
C'2.0P'*R'*K =
P!R',A',C'!
E!
```

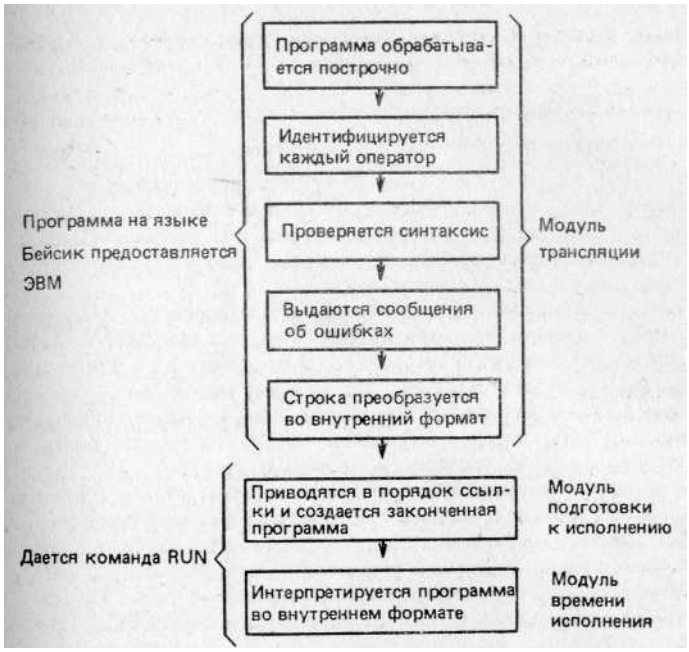


Рис. 9.2. Стадии процесса трансляции и исполнения программы

После того как дана команда RUN, внутренняя форма может интерпретироваться системой времени исполнения. По своей структуре внутренний формат близок к исходной программе на Бейсике и может быть без особых затруднений отредактирован или переведен обратно в текст на Бейсике. При преобразовании во внутренний формат каждый оператор программы на Бейсике тщательно проверяется на наличие ошибок. И хотя скорость исполнения программы при таком подходе ниже, чем в случае описанной выше чистой компиляции, она намного превышает ту, которую обеспечивает чистая интерпретация.

Компилятор состоит из нескольких различных частей. Первой и самой большой из них является система трансляции операторов Бейсика во внутренний формат. Вторая вступает в действие, когда дана команда RUN, и приводит в порядок ссылки и прочие 'недоделки", оставленные первой системой, работающей по принципу построчной обработки. Она называется системой подготовки к исполнению. Третьей частью является система времени исполнения, интерпретирующая внутренний формат во время исполнения программы. Другие части компилятора занимаются обратным преобразованием внутреннего формата в операторы Бейсика при выполнении команд LIST или SAVE, интерпретацией немедленно исполняемых операторов Бейсика и выполнением команд Бейсика. Таким образом, из простого транслятора операторов Бейсика вырастает цельная система, обеспечивающая работу в режиме диалога.

Многие системы с Бейсиком, включая некоторые из тех, которые разработаны для микроЭВМ, не содержат системы трансляции, обеспечивающей проверку операторов программы по мере их набора, и выполняют эту проверку после того, как дана команда RUN. Вместо немедленного выявления синтаксических ошибок в одном операторе они

дают перечень ошибок для всех операторов сразу. Ясно, что это не лучший способ реализации диалоговой системы программирования.

9.5. ОПЕРАЦИОННЫЕ СИСТЕМЫ

(При первом чтении этот раздел лучше пропустить)

Операционной системой называется программа, управляющая работой ЭВМ. Либо вся ОС, либо ее часть постоянно находится в памяти ЭВМ во время исполнения других программ. ОС создает среду,

в которой может работать остальное программное обеспечение, включая программы пользователя. В результате она отделяет пользователя от утомительных деталей работы с аппаратной частью ЭВМ.
Если ЭВМ достаточно мощна и к ней подсоединено много терминалов, то ОС обеспечивает режим разделения времени. Иначе говоря, она выделяет порции машинного времени каждому терминалу, за которым работает пользователь. В результате ее работы у пользователя создается впечатление, что он имеет монопольный доступ к ресурсам ЭВМ. Скорее всего, единственным заметным для него эффектом присутствия других пользователей будет довольно существенное замедление реакции системы, если большинство терминалов (ВТУ) одновременно занято работой.
Другой задачей современной операционной системы является ведение файловой системы. Понятие файловой системы охватывает допустимые типы файлов, форматы хранения данных на лентах и дисках, соглашения об именах файлов и способы доступа к файлам из программ. ОС снабжается различными вспомогательными программами для манипулирования файлами, например для копирования, удаления, переименования, распечатывания и редактирования. Иногда можно редактировать на ВТУ только файлы определенных типов, называемых файлами в графическом и терминальном формате. В действительности, эти файлы хранятся в том виде, в каком они набираются на клавиатуре; файлы других типов предназначены исключительно для внутреннего применения, т. е. для использования программами, и распечатывание таких файлов на ВТУ не будет иметь особого смысла.
Многие микроЭВМ с Бейсиком не имеют отдельной операционной системы. Они рассчитаны на единственное применение, а именно выполнение написанных на Бейсике программ, и многие функции, требуемые от операционной системы, встроены в систему с Бейсиком. Примером такого подхода служит персональная ЭВМ PET фирмы Commodore. При включении питания этой ЭВМ в нее автоматически загружается система с Бейсиком, обеспечивающая выполнение всех операций, требуемых для создания и исполнения программы.
При работе с PET можно выйти из системы с Бейсиком (с помощью команды SYS 64785) и оказаться в простой операционной системе, называемой монитором. Она обеспечивает выполнение очень ограниченного числа операций для работы с программами, написанными в машинных кодах. Такие системы с Бейсиком, иллюстрируемые рис. 9.3, а, являются типичными представителями специализированного программного обеспечения ЭВМ, в данном случае рассчитанного на исполнение программ, написанных на Бейсике. Так устроены многие системы для микроЭВМ; они загружаются в "голую" машину либо вообще не имеющую монитора, либо имеющую очень простой монитор. Эти системы способны сами управляться с наиболее элементарными операциями, выполненными аппаратной частью ЭВМ.
Одной из широко распространенных операционных систем, предназначенной для микроЭВМ и совершенно отдельной от системы с Бейсиком, является CP/M. В подразд 8.5.2 описана процедура загрузки, согласно которой CP/M требует загрузить до загрузки Бейсика. Перейдем теперь к обсуждению некоторых деталей процесса запуска программы. Перед исполнением программа должна быть загружена в память операционной

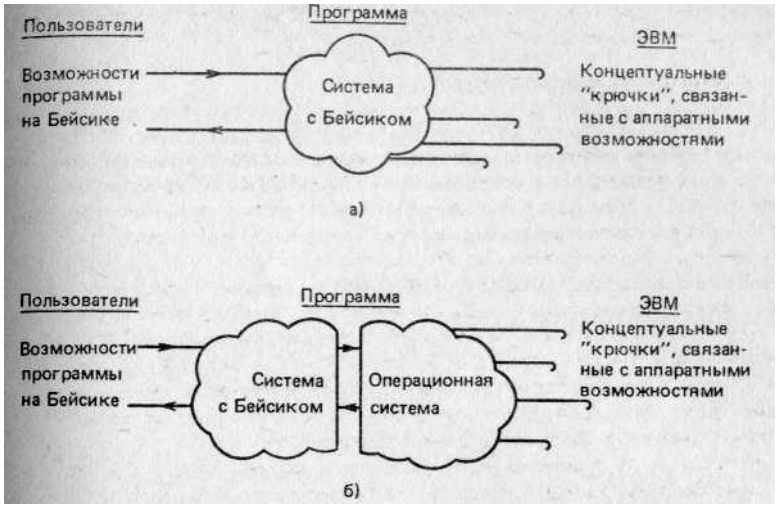


Рис. 9.3. Операционные системы с Бейсиком:

а - встроенный Бейсик (например, в системах PET и ZX 81 фирмы Sinclair); *б* - отдельный Бейсик (например, для операционной системы CP/M) системой или эквивалентной частью системы с Бейсиком, если последняя функционирует сама по себе. Но каким же образом загружается эта начальная система, способная загружать другие программы? Решение этой проблемы обеспечивается очень простой программой, написанной в машинных кодах и называемой загрузчиком. Первые модели ЭВМ 50-х годов имели ряд тумблеров, посредством которых оператор должен был вручную заносить в память начальный загрузчик, но у современных микроЭВМ он, к счастью, встроен в специальное ПЗУ, и включение питания активизирует начальный загрузчик, который инициирует загрузку всей системы.

Первые операционные системы в основном требовались для того, чтобы избежать подобной загрузки в ЭВМ каждой отдельной программы. Быстро развиваясь, они скоро стали обеспечивать пакетный режим работы, при котором машине предоставлялось сразу несколько программ. Когда одна из них завершалась, управление передавалось операционной системе, способной после этого загрузить и исполнить следующую программу.

Постепенно функции операционных систем усложнялись. Разрабатывались стандартные процедуры для управления работой устройств ввода и вывода, а также для устранения простых ошибок. Появились языки управления заданиями в виде команд, обеспечивающих выбор требуемого режима исполнения любой конкретной программы. Когда к ЭВМ стали подключать терминалы, работающие в режиме непосредственного доступа, то языки управления заданиями развились в более богатые системы команд, в которых используются преимущества диалоговой природы взаимодействия между пользователем и ЭВМ.

Современные операционные системы рассчитаны на то, чтобы дополнять аппаратные средства, вместе с которыми они предоставляют пользователю то, что в действительности является виртуальной машиной. Этим термином называют систему, обладающую Расширенными возможностями по сравнению с теми, которые фактически реализуются

аппаратными средствами. Например, если аппаратные средства позволяют вводить и выводить только по одному символу за раз, то операционная система может предусмотреть буферизацию и организовать работу так, чтобы программы при необходимости могли считывать или выводить целую строку символов.

Каждый тип ЭВМ требует, чтобы операционная система разрабатывалась с учетом свойств аппаратных средств, но можно взять операционную систему общего назначения например CP/M, и адаптировать ее для различных ЭВМ. Действия пользователя при работе на различных ЭВМ с

операционной системой CP/M будут почти одни и те же, и в этом случае программы и другие программные средства, например, системы с Бейсиком, можно будет легко переносить от одной модели ЭВМ к другой.

ПРИЛОЖЕНИЕ I. РЕШЕНИЕ УПРАЖНЕНИЙ РЕШЕНИЕ УПРАЖНЕНИЙ К ГЛ. 1

1.1 (а) Z=6, B=2, A=2
(б) A=-5, B=5, Q=$-5^1/_2$
(в) B=0,A=$^1/_2$, Z=8
(г) A=110

1.2 (а) LET A=B+C*C
(б) LET A=(A+B)/C
(в) LET Y = (X+A)/(Y-B)
(г) LET Z = 1/(1 -1/(A+B))

1.3 Эта программа присваивает различные значения переменным A, B, C и D. Затем она присваивает сумму этих значений, а потом среднее значение переменной Z, после чего печатает среднее значение в строке 70.

1.4

```
10 INPUT A
20 INPUT B
30 INPUT C
40 INPUT D
50 Z=A+B+C+D
60 Z=Z/4
70 PRINT Z
80 END
```

или

```
10 INPUT A,B,C,D
50 Z=A+B+C+D
60 Z=Z/4
70 PRINT Z
80 END
```

1.5

```
10  INPUT M,T,R
20  I=M*T*(R/100)
30  PRINT I
40  END
```

или

```
10 PRINT "ПОЖАЛУЙСТА, ВВЕДИТЕ СУММУ, СРОК, ПРОЦЕНТ"
20 INPUT M,T,R
30 I=M*T*(R/100)
40 PRINT "ПРИ ПРОЦЕНТНОЙ СТАВКЕ";R;"% ЗА";T;"ГОДА"
50 PRINT "НА СУММУ";M;"ФУНТОВ СТЕРЛИНГОВ БУДЕТ НАЧИСЛЕНО"
60 PRINT I;"ФУНТОВ СТЕРЛИНГОВ ПРОЦЕНТОВ"
70 END
```


РЕШЕНИЕ УПРАЖНЕНИЙ К ГЛ. 2

2.1

```
10 INPUT M, T, I
20 R=1+I/100
```

```
30 P=M*R**T*(R-l)/(R**T-1)
40 PRINT "ДЛЯ ПОГАШЕНИЯ ССУДЫ В";M;", ВЫДАННОЙ"
50 PRINT "НА";T;"ГОДА С ПРОЦЕНТНОЙ СТАВКОЙ";I;"%"
60 PRINT "ТРЕБУЕТСЯ ЕЖЕГОДНО ВЫПЛАЧИВАТЬ";P
70 END
```

2.2

```
10 INPUT L,H, W
20 PRINT "КОРОБКА ДЛИНОЙ";L;"ВЫСОТОЙ";H;"ШИРИНОЙ";W
30 REM ВЫЧИСЛЕНИЕ ОБЪЕМА
40 V=L*H*W
50 REM ВЫЧИСЛЕНИЕ ПЛОЩАДИ ПОВЕРХНОСТИ
60 S=2*(W*H+H*L+W*L)
70 REM ВЫЧИСЛЕНИЕ СТОИМОСТИ В ПЕНСАХ
80 C = 5+S*0.02
90 REM
100 PRINT "ОБЪЕМ =";V
110 PRINT "ПЛОЩАДЬ ПОВЕРХНОСТИ =";S 120 PRINT "СТОИМОСТЬ (В ПЕНСАХ) =";C 130 END
```

2.3 Добавьте к программе из упражнения 2.2 оператор 85 C=C+(S- W*L) *0.005

2.4

```
10 PRINT "КРЕДИТНАЯ ФИРМА TAXHAVEN"
20 PRINT "---------------------------"
30 PRINT
40 REM B - БАЛАНС 50 I=B*8/100
60 PRINT "                    ПРЕДЫДУЩИЙ БАЛАНС";B
70 PRINT "                                ПРОЦЕНТЫ";I
80 PRINT
90 PRINT "ТОВАР                 СТОИМОСТЬ"
100 PRINT "-----                                ----------"
110 T=B+1
120 FOR J=1 TO 3
130   INPUT A$,C
140   PRINT A$,C
150   T=T+C
160 NEXT J
170 PRINT "                                                ----"
180 PRINT "                                ИТОГО       ":T
190 PRINT "                                                ----"
200 PRINT
210 PRINT "МИНИМАЛЬНАЯ ВЫПЛАТА";T/ 10
220 END
```

Строки 120 - 160 можно заменить на

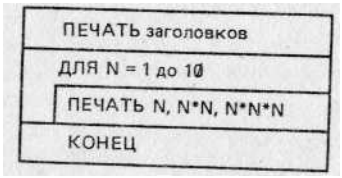
```
120   INPUT A$,C
124   PRINT A$,C
126   T=T+C
130   INPUT A$,C
134   PRINT A$,C
136   T=T+C
```

```
140 INPUT A$,C
144 PRINT A$,C
146 T=T+C
```
Они обеспечивают то же действие, что и цикл, организация которого детально обсуждается в гл. 3.

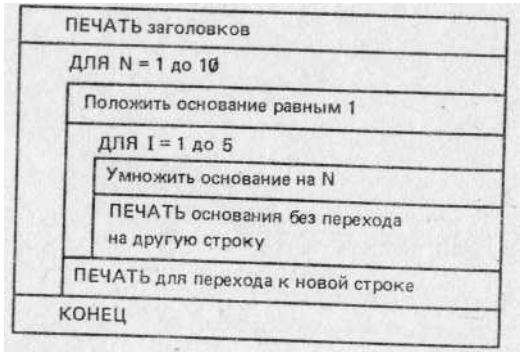
РЕШЕНИЯ УПРАЖНЕНИЙ К ГЛ. 3

3.1

```
10 INPUT M
20 C=25
30 IF M<=2 THEN 50
40   C=C+2*(M-2)*8
50 PRINT C
60 END
```
Для решения упражнения при дополнительном условии замените строку 40 на оператор
```
40   C=C + INT(2*(M-2 + .49))*8
```
3.2




```
10  PRINT   "ЧИСЛО" , "КВАДРАТ" , "КУБ"
20   FOR   I=1   TO   10
30        PRINT  N,N*N, N*N*N
40   NEXT N
50   END
```
3.3

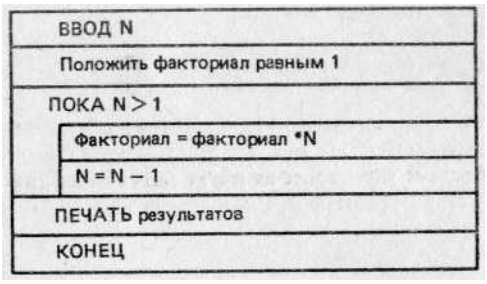



```
10 PRINT "СТЕПЕНИ N"
20 PRINT "---------"
30 PRINT
40 FOR N=1 TO 10
300
```

```
50       S=1
60       FOR   I=1   TO   5
70           S=S*N
80         PRINT  S;
90       NEXT I
100        PRINT
```

```
110   NEXT  N
120   END
```

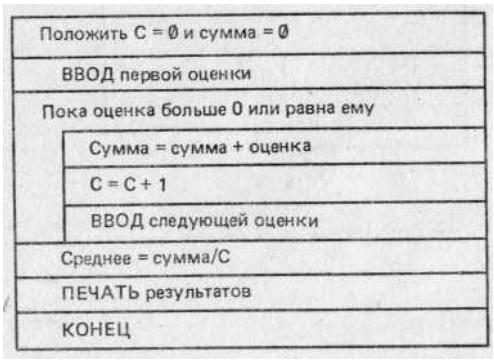
3.4




```
10   INPUT N
20   F=1
30   IF N<=1 THEN 70
40   F=F*N
50   N=N-1
60   GOTO 30
70   PRINT "ФАКТОРИАЛ РАВЕН";F
80   END
```

Обратите внимание на то, как должен преобразовываться цикл ПОКА в операторы IF и GOTO. В приведенной выше программе значение N не сохраняется; если в дальнейшем оно может потребоваться, то его надо скопировать в другую переменную (скажем, M). Эта программа при отрицательных N будет изображать (ошибочно) F=1; поэтому надо принять соответствующие меры, если такие значения N возможны.

3.5



301

```
10 REM ВВОД ОЦЕНОК. ПРИЗНАК КОНЦА ВВОДА - ОТРИЦАТЕЛЬНОЕ ЧИСЛО
20 REM C=СЧЕТЧИК ЧИСЛА ВВЕДЕННЫХ ОЦЕНОК
30 REM S=СУММА ВВЕДЕННЫХ ОЦЕНОК
40 C=0
50 S=0
60   INPUT M
70   IF M<0 THEN 120
80        S=S+M
90        C=C+1
100    INPUT M
110   GOTO   70                                    .
120   A=S/C
```

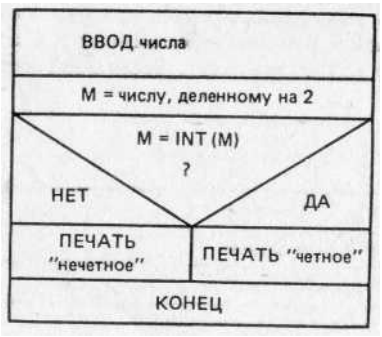
```
130  PRINT  "СРЕДНЯЯ ОЦЕНКА РАВНА";A
140  END
```

Обратите внимание на преобразование цикла ПОКА в операторы IF и GOTO. Перевод ПОКА в операторы GOTO-IF требует, чтобы цикл начинался с оператора IF. Произвольные передачи управления не могут быть описаны посредством структограмм, поэтому в структурном программировании модификация вида

```
50  .....
60  INPUT M
70  IF M<0 THEN 110
80    S=S+M
90    C=C+1
100  GOTO 60
110  .....
```

не допускается, хотя и не выглядит ошибочной. Ее надо избегать потому, что при применении подобных приемов в более сложной ситуации легко допустить ошибку.

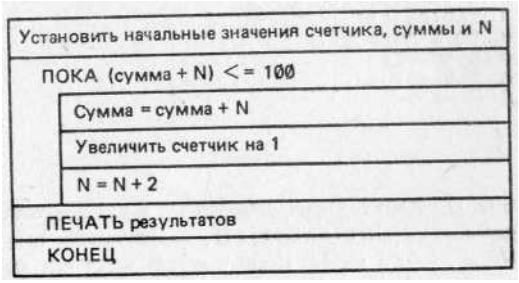
3.6




```
10  REM  ТЕСТ  НА  ЧЕТ  ИЛИ  НЕЧЕТ
20 INPUT N
30  M=N/2
40  REM
50  IF M=INT(M)  THEN  80
60      PRINT  "ЧИСЛО";N;"НЕЧЕТНОЕ"
70  GOTO  90
80      PRINT  "ЧИСЛО";N;"ЧЕТНОЕ" 90  END
```

302

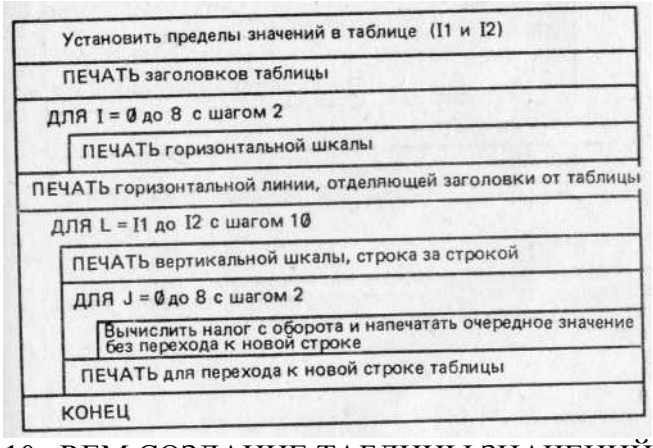
3.7




```
10  REM  СУММА  ЧЕТНЫХ  ЧИСЕЛ
20  REM  C  -  СЧЕТЧИК,  S  -  СУММА.  N  -  ТЕКУЩЕЕ  ЧИСЛО
30  C=1
```

```
40   S=0
50   N=2
60 IF (S+N)>100 THEN 110
70   S=S+N
80   C=C+1
90   N=N+2 100 GOTO 60
110 PRINT "В СУММУ";S;"ВОШЛО";C-1;"ЗНАЧЕНИЙ" 120 END
```

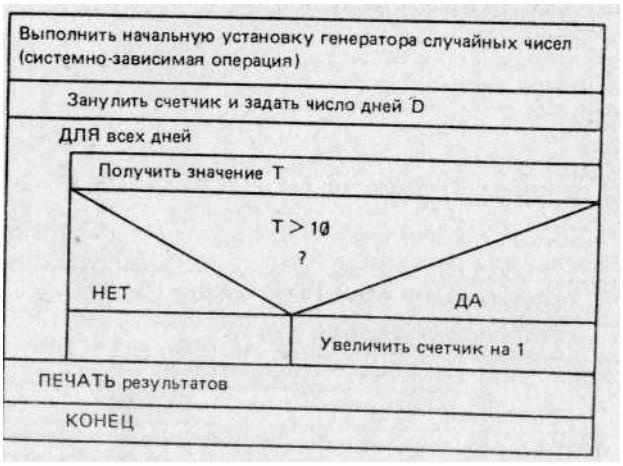
3.8



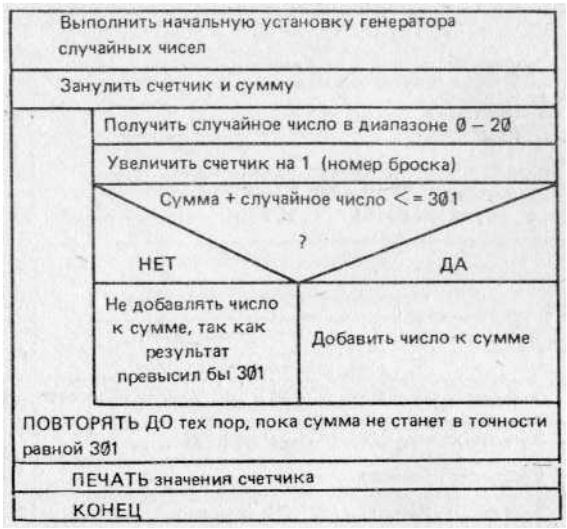

```
10   REM СОЗДАНИЕ ТАБЛИЦЫ ЗНАЧЕНИЙ НАЛОГА С ОБОРОТА
20   REM НАЛОГ СОСТАВЛЯЕТ 15%
30   I1=0
50   PRINT "НАЛОГ С ОБОРОТА ДЛЯ ЦЕН ОТ" ; I1 ; "ДО" ; I2+8 ; "ПЕНСОВ"
60   PRINT "-------------------------------------
70   PRINT
80   FOR I=0 TO 8 STEP 2
90   PRINT TAB(10+I*3);I;
303



100 NEXT I
110 PRINT "----------------------------------------"
120 FOR L=I1 TO I2 STEP 10
130   PRINT L;TAB(6) : "I" ;
140   FOR J=0 TO 8 STEP 2
150     V=INT(0.15*(L+J))
160     PRINT TAB(10+J*3);V;
170   NEXT J
180   PRINT
190 NEXT L
200 END
```

3.9

В предположении, что RND возвращает результат в диапазоне 0 ... 0.9999999, рассмотрите выражение T= INT(RND*15 +1), результатами вычисления которого являются целые числа от 1 до 15. Первые десять чисел поставьте в соответствие преждевременному отходу поезда; посадка на поезд возможна только при T>10,




```
10 НЕМ МОДЕЛИРОВАНИЕ ПОСАДКИ НА ПОЕЗД
2 0 REM НАЧАЛЬНАЯ УСТАНОВКА ГЕНЕРАТОРА СЛУЧАЙНЫХ ЧИСЕЛ
30 INPUT "ПОЖАЛУЙСТА. ВВЕДИТЕ ЧИСЛО";N
40 RANDOMIZE N
50 REM
60 C=0
70 D=20
80 FOR I=1 TO D
90   T=INT(RND*15+1)
100   IF T>10 THEN C=C+1
110 NEXT I
120 PRINT "ВЫ УСПЕЕТЕ НА ПОЕЗД В";C;"СЛУЧАЯХ" 130
END
304
```

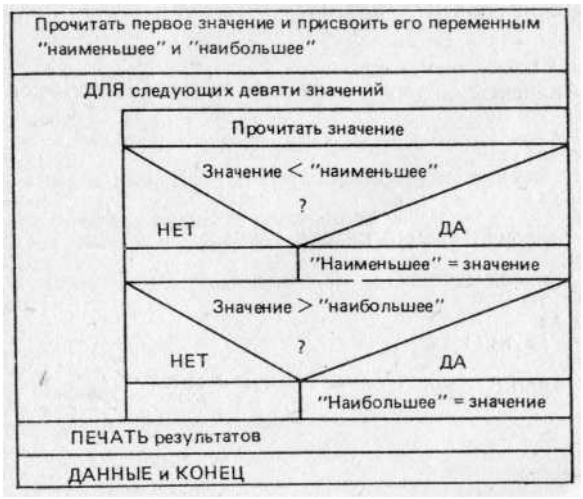



```
10   REM МОДЕЛИРОВАНИЕ ИГРЫ В МЕТАНИЕ СТРЕЛОК
20INPUT "ПОЖАЛУЙСТА, ВВЕДИТЕ ЧИСЛО";N
30   RANDOMIZE N
40 S=0
50   C=0
60       L=INT(21*RND)
70       C=C+1
```

```
80      IF  (S+L)<=301  THEN  S=S+L
90  IF  S<>301  THEN  60
100  PRINT  "ПОЛУЧЕНО  301  ОЧКО  ПОСЛЕ";C;"БРОСКОВ"
110  END
```

РЕШЕНИЯ УПРАЖНЕНИЙ К ГЛ. 4

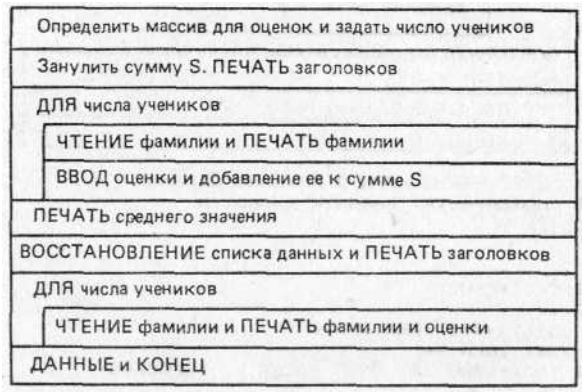
4.1



305

```
10  REM НАИМЕНЬШЕЕ И НАИБОЛЬШЕЕ
20  READ T
30  S=T
40  L=T
50  FOR I=1 TO 9
60  READ T
70  IF T<S THEN S=T
80  IF T>L THEN L=T
90  NEXT I
100  PRINT "НАИМЕНЬШЕЕ ЗНАЧЕНИЕ РАВНО";S
110  PRINT "НАИБОЛЬШЕЕ ЗНАЧЕНИЕ РАВНО" ;L
120  DATA 10,0,19,20,7,39,267,4,1,500
130  END
```

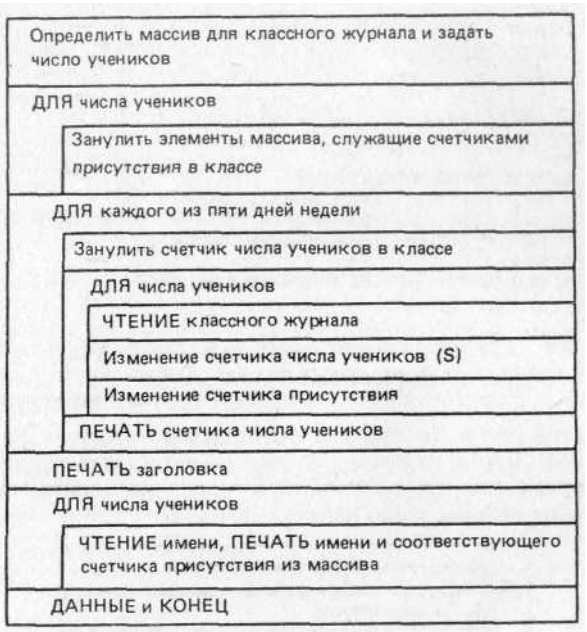
4.2




```
10  REM  КЛАССНЫЙ  ЖУРНАЛ  (ДЛЯ  ПЯТИ  УЧЕНИКОВ)
20  DIM  M(5)
```

```
30   N=5
40 S=0
50 PRINT "ВВЕДИТЕ ОЦЕНКУ КАЖДОГО ИЗ СЛЕДУЮЩИХ УЧЕНИКОВ:
60 FOR I=1 TO N
70   READ A$
80   PRINT A$;
90   INPUT M(I)
100   S=S+M(I)
110 NEXT I
120 PRINT "СРЕДНЯЯ ОЦЕНКА КЛАССА РАВНА";S/N
130 RESTORE
140 PRINT "СПИСОК ОЦЕНОК:"
150 FOR I=1 TO N
160   READ A$
170   PRINT A$,M(I)
180 NEXT I
190 DATA "ДЖОАН","ЛАРА";"САРА";"ДОУН";"ЭСТЕР"
200 END
306
```

4.3




```
10   REM ЖУРНАЛ ПОСЕЩАЕМОСТИ КЛАССА (ДЛЯ ТРЕХ УЧЕНИКОВ)
20   DIM N(3)
30   M=3
40   FOR I=1 TO M
50   N(I)=0
60   NEXT I
70   FOR D=l TO 5
80   S=0
90   FOR I=1 TO M 100     READ P 110     5=S+P 120     N(I)=N(I)+P 130   NEXT I
140   PRINT "В ДЕНЬ" ;D; "ПРИСУТСТВОВАЛО" ;S; "УЧЕНИКОВ"
```

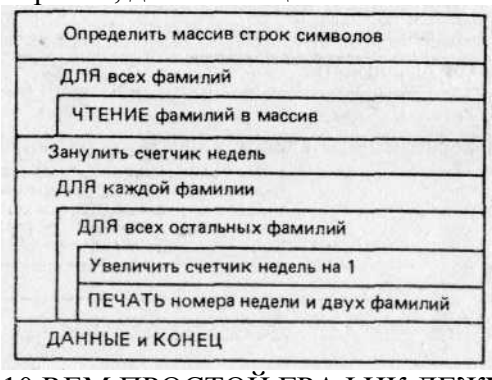
```
150   NEXT D
160   PRINT "ИНДИВИДУАЛЬНАЯ ПОСЕЩАЕМОСТЬ"
170   FOR I=1 TO M 180   READ A$
190   PRINT A*;" ПРИСУТСТВОВАЛ";N(I);"ДНЯ"
200   NEXT I
210   DATA 0,1,1
220   DATA 1,1,1
230   DATA 1,1,1
240   DATA 1,0,0
250   DATA 1,0,1
260   DATA "ПИТЕР", "ДЖОН", "ДЭВИД"
270   END
307
```

4.4

```
10 REM ОБРАБОТКА СТРОК СИМВОЛОВ (С ПОМОЩЬЮ ФУНКЦИЙ
15 REM БЕЙСИКА MICROSOFT)
20 A$="ANN"
30 B$="KENNY"
40 C$=A$+" "+B$
50 PRINT C$
60 REM
70 PRINT LEFT$(A$, 1) ; " " ; LEFT$ (B$ , 1)
80 REM
90 D$="M$"
100 E$="201 TOWNEND ROAD. CREWE"
110 PRINT D$;" ";C$          :REM ПЕЧАТЬ ФАМИЛИИ
120 J=INSTR(1, E$,",")      :REM НАЙТИ ПОЗИЦИЮ ЗАПЯТОЙ В АДРЕСЕ
130 PRINT LEFT$(E$,J)      :REM ПЕЧАТЬ УЛИЦЫ
140 J=LEN(E$)-J
150 PRINT RIGHT$(E$,J)    :REM ПЕЧАТЬ ГОРОДА
160 END
```

Строки 100 - 150 иллюстрируют расщепление строки с адресом для печати города и улицы, как обычно, на отдельных строках. Для этого находится положение запятой в строке и части строки слева и справа от запятой печатаются в разных строках.

4.5 Разработка и программа основываются на приеме, согласно которому берется первая фамилия, и к ней по одной подставляются все остальные фамилии. Затем берется вторая фамилия и т. д. Таким образом, для воплощения этого плана удобно использовать вложенную пару циклов FOR-NEXT.




```
10 REM ПРОСТОЙ ГРАФИК ДЕЖУРСТВ (НА ПРИМЕРЕ ПЯТИ ФАМИЛИЙ)
```

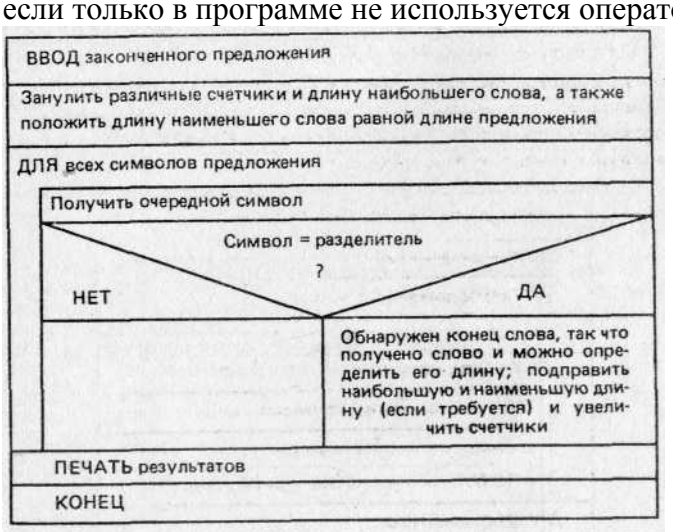
```
20 DIM N$(10)
30 N=5
40 FOR I=1 TO N
50   READ N$(I)
60 NEXT I
70 W=0
80 FOR I=1 TO N
90   FOR J=I+1 TO N
100     W=W+1
120     PRINT "НА НЕДЕЛЕ";W;"ДЕЖУРЯТ";N$(I);" И ";NS(J)
130   NEXT J 140 NEXT I
150 DATA NAME1,NAME2,NAME3,MAME4,NAME5
160 END
308
```

В строке 150 можно было бы подставлять настоящие фамилии, но выбранные в программе хорошо иллюстрируют систему составления графика дежурств, выдаваемого программой после запуска командой RUN.

Для разработки простой программы, учитывающей требование, чтобы никакая фамилия не появилась в списке более двух недель подряд, необходимо подумать. Эта программа имеет ту же форму, что и приведенная выше, но совсем по-другому выбирает фамилии. Модифицируйте следующие строки:

```
10    REM МОДИФИКАЦИЯ ГРАФИКА ДЕЖУРСТВ (НЕ БОЛЕЕ ДВУХ НЕДЕЛЬ
11    REM ПОДРЯД ОДНОМУ ЧЕЛОВЕКУ)
90   FOR I=1 TO N-1
100      FOR J=1 TO N-1
120         PRINT "НА НЕДЕЛЕ"; W; "ДЕЖУРНЫЕ"; N$(J); " И "; N$(J+1)
```

4.6 При решении предполагается, что слова должны отделяться друг от друга либо I запятой, либо точкой с запятой, либо точкой или пробелом. При вводе предложение 1 надо окаймлять кавычками, если только в программе не используется оператор LINPUT или ему подобный.

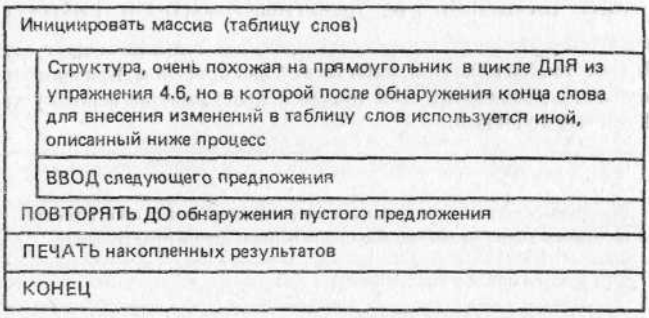



```
10   REM АНАЛИЗ ПРЕДЛОЖЕНИЯ (С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ
15   REM БЕЙСИКА MICROSOFT)
2в   INPUT SI
30   S=LEN(S$)
40   S0=S
```

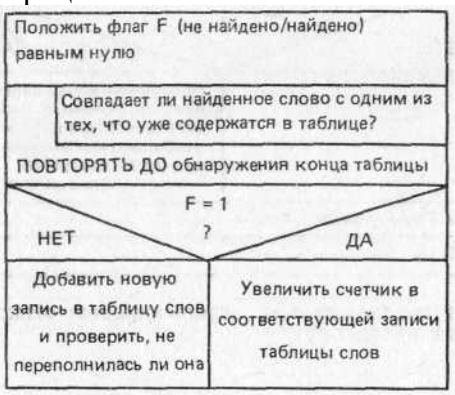
```
50  L0=0
60  J=l          :REM УКАЗАТЕЛЬ НА НАЧАЛО СЛОВА
70  T=0
80  W=0          :REM СЧЕТЧИК СЛОВ
90  FOR I=1 TO S
100       A$=MID$(S$,I,1)
110      IF A$ = " " THEN 160
120       IF A$ = "." THEN 160
130       IF A$ = ";" THEN 160
140        IF A$ = " , " THEN 160
150       GOTO 220
309
```

```
160  L=LEN(MID$(S$,J,I-J)) 170   W=W+1
180  T=T+L
190  J=I+1
200  IF L<S0 THEN S0=L
210  IF L>L0 THEN L0=L
220  NEXT I
230  PRINT "МАКСИМАЛЬНАЯ ДЛИНА СЛОВА. РАВНА" ;L0
240  PRINT "МИНИМАЛЬНАЯ ДЛИНА СЛОВА РАВНА" ;S0
250  PRINT "СРЕДНЯЯ ДЛИНА СЛОВА РАВНА";T/W
260  END
```

4.7



Процесс отыскания слова:




```
10 REM ПРОГРАММА ОПРЕДЕЛЕНИЯ ЧАСТОТЫ СЛОВ В ТЕКСТЕ
15 НЕМ ИЗ НЕСКОЛЬКИХ ПРЕДЛОЖЕНИЙ
20 REM (ДЛЯ ПРИМЕРА МАССИВАМ ДАНЫ РАЗМЕРЫ 20)
30 DIM W$(20),W(20)
```

```
40 FOR I=1 TO 20
50   W$(I)=""        :REM БУДЕТ СОДЕРЖАТЬ СЛОВА
60   W(I)=0                  :REM БУДЕТ СОДЕРЖАТЬ СООТВЕТСТВУЮЩИЕ ЧАСТОТЫ
70 NEXT I
80 W0=l                       :REM УКАЗАТЕЛЬ НА НАЧАЛО ТАБЛИЦЫ СЛОВ
90 INPUT S$                   :REM ПОЛУЧИТЬ ПЕРВОЕ ПРЕДЛОЖЕНИЕ ИЛИ ФРАЗУ
310



100      S=LEN(S$)
110      J=l
120      FOR I=1 TO S
130          AS=MID$(S$, I ,1)
140          IF AS=" " THEN 190
150          IF A$="." THEN 190
160          IF A$=" , " THEN 190
170          IF A$=";" THEN 190
180          GOTO 340
190             B$=MID$(S$, J, I-J)
200              J=I+1
210             F=0
220             K=1
230                 IF B$=W$(K) THEN F=1
240                   K=K+1
250                IF F=0 AND K<W0 THEN 230
260               IF F=1 THEN 330
270                   W$(W0)=B$
280                   W(W0)=1
290                   W0=W0+1
300                    IF W0<=20 THEN 340
310                       PRINT "ТАБЛИЦА СЛОВ ПЕРЕПОЛНЕНА"
320                       STOP
330                   W(K-1)=W(K-1)+1
340       NEXT I
350        INPUT S$
360    IF S$<>"" THEN 100
370   REM
380   REM СЛОВА НАХОДЯТСЯ В W$( ) , А ИХ СЧЕТЧИКИ - В W( ),
390   REM ДЛЯ СОРТИРОВКИ СЛОВ ПО АЛФАВИТУ ПРИМЕНИТЕ СЛЕДУЮЩУЮ
400   REM ПРОЦЕДУРУ (НЕ ЗАБУДЬТЕ ОТРАЗИТЬ ВСЕ ПЕРЕСТАНОВКИ
410   REM И В МАССИВЕ W$( ))
420   REM
430 PRINT "СЛОВО" , "ЧАСТОТА"
440 FOR I=1 TO W0-1
450   PRINT W$(I),W(I)
460 NEXT I
470 END
```

4.8

Для простоты предположим, что данные никак не упорядочены, и для поиска используем простой метод полного перебора. Программа допускает много усовершенствований; в качестве примера того, что можно включить в нее, ниже выбран прием повторения поиска с укорачиваемым образцом. Это позволяет обнаружить искомую запись даже в том случае, если при составлении запроса на поиск была допущена какая-либо опечатка.

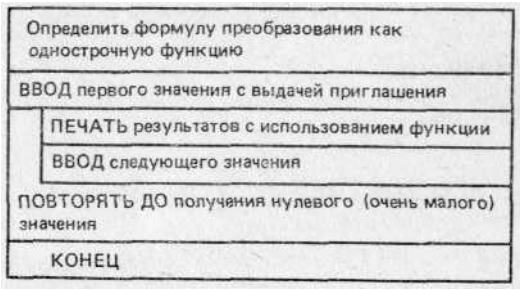
311


```
10  REM  ПОИСК  ДАННЫХ
20  INPUT  "ИНТЕРЕСУЮЩАЯ  ВАС  ТЕМА";Q$
30  P=LEN(Q$)
40 N=3        :REM ЧИСЛО ЭЛЕМЕНТОВ ДАННЫХ
50 REM        НА ПРАКТИКЕ ОНО МОЖЕТ ИСЧИСЛЯТЬСЯ СОТНЯМИ
б0 F=0
70 I=1
80 READ A$
90 IF LEFT$(A$,P)=LEFT$(Q$,P) THEN F=1
100     I=I+1
110  IF F=0 AND I<=N THEN 80
120  IF F=1 THEN 170
130    PRINT "ТЕМА "; LEFT$ (Q$, P);" НЕ НАЙДЕНА"
140    PRINT "ПОВТОР С УКОРОЧЕННЫМ НАЗВАНИЕМ"
150    RESTORE
160    P=P-1
170 IF F=0 AND P>=1 THEN 70
180 REM
190 IF F=1 THEN 230
200  PRINT "ТЕМА НЕ НАЙДЕНА"
210  STOP
230 PRINT "ОБНАРУЖЕНО СОВПАДЕНИЕ С "; LEFT$(Q$,P)
```

```
240 PRINT "-----------------------------------"
250 PRINT "ПОЛНОЕ НАЗВАНИЕ НАЙДЕННОЙ ТЕМЫ: ";AS
260 REM
270 DATA CHATTEL-ДВИЖИМОЕ ИМУЩЕСТВО (СУЩЕСТВИТЕЛЬНОЕ)
280 DATA CHEAP-ИМЕЮЩИЙ НЕБОЛЬШУЮ ЦЕНУ (ПРИЛАГАТЕЛЬНОЕ)
290 DATA CHERISH-ЛЕЛЕЯТЬ (ГЛАГОЛ)
300   END
312
```

РЕШЕНИЯ УПРАЖНЕНИЙ К ГЛ. 5

5.1




```
10 REM ОПРЕДЕЛИТЬ ФУНКЦИЮ С ПОМОЩЬЮ ОПЕРАТОРА FN И
15 REM ВЫЗВАТЬ ЕЕ ДЛЯ ПРЕОБРАЗОВАНИЯ ТЕМПЕРАТУРЫ ИЗ
17 REM ШКАЛЫ ЦЕЛЬСИЯ В ГРАДУСЫ ФАРЕНГЕЙТА
20 DEF FNC(X)=32+180*X/100
30 REM
40 INPUT "ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ"; C
50   PRINT "ТЕМПЕРАТУРА ПО ФАРЕНГЕЙТУ =";FNC(C)
60   INPUT "ТЕМПЕРАТУРА ПО ЦЕЛЬСИЮ"";C .
70 IF ABS(C).GT.1.0E-7 THEN 50
80 END
```

5.2 Предлагаемая в качестве решения программа составляет каждое письмо по следующей форме: в правом верхнем углу помещаются Ваша фамилия, адрес и дата; ниже в левой строке листа помещаются фамилия и адрес получателя; непосредственно под этой строкой со слов "Дорогой..." начинается текст письма, завершаемый оборотом "Искренне Ваш" и достаточным количеством пустых строк, образующих промежуток для подписи перед началом нового письма. Предполагается, что текст письма хранится в операторах DATA, а фамилии и адреса вводятся с терминала. На практике одна или обе из этих совокупностей данных должны были бы браться из файла.

Уровень 1

Ввести всю стандартную информацию, например дату и наименование продукции,
описываемой в письме. Ввести набор фамилий и адресов. ПОКА не все фамилии обработаны,
скомпоновать и напечатать письмо. КОНЕЦ цикла.

Детализация 1.1. Ввести всю стандартную информацию

Напечатать приглашения к вводу и ввести дату, название продукции (при необходимости) и другую требуемую информацию.

Детализация 1.2. Ввести набор фамилий и адресов

Напечатать приглашение к вводу числа фамилий и ввести его в переменную N.
ДЛЯ N элементов данных ввести фамилию, ввести три строки адреса (город, район, улица)
КОНЕЦ цикла

Детализация 1.3. Скомпоновать и напечатать письмо
Напечатать у правого края листа Вашу фамилию и адрес и проставить под ними дату Напечатать у левого края листа фамилию и адрес получателя, а под ними - "Дорогой (фамилия)!"
ДЛЯ числа строк в тексте
ЧТЕНИЕ строки текста,
ПЕЧАТЬ этой строки. КОНЕЦ цикла.
Напечатать у правого края листа "Искренне Ваш". Напечатать достаточное число простых строк.
Приведенных выше описаний вполне достаточно для формирования уровня 2, но на практике число фамилий и адресов, вводимых согласно детализации 1.2, будет ограничено размерами памяти ЭВМ, поэтому установим ограничение на число повторений основной части программы равным, скажем, 20 за один прием. В общих чертах уровень 2 приобретет следующий вид:
Уровень 2
(1.1) для определенных данных. ПОВТОРЯТЬ для наборов фамилий и адресов:
ПОКА не все фамилии обработаны, (1.3) для письма.
КОНЕЦ цикла.
Спросить, будут ли еще наборы данных? До тех пор, пока не будет дан отрицательный ответ.
Теперь основным моментом, требующим детализации, является печатание письма. Действия ЧТЕНИЕ и ПЕЧАТЬ строки выполняются, по крайней мере, дважды (подлежат обработке Ваш адрес и текст письма), поэтому их надо раскрыть.
Детализация 2.1
Получить начальную позицию L и число повторений К. ДЛЯ К повторений
ЧТЕНИЕ строки,
ПЕЧАТЬ строки, начиная с L-й позиции листа. КОНЕЦ цикла.
Полезно также иметь блок для вывода пустых строк.
Детализация 2.2
Получить число пустых строк К. ДЛЯ К повторений
ПЕЧАТЬ. КОНЕЦ цикла.
Обратите внимание на то, что описанные выше действия уровня 2 можно было бы с равным успехом изобразить в виде структограммы. Теперь можно было бы описать или изобразить в виде структограмм заключительный уровень разработки, на кото-

ром принимаются решения относительно имен переменных, счетчиков циклов и т. д. Окончательный результат представляет собой следующую программу:

```
10 REM ПРОГРАММА ДЛЯ ПЕЧАТИ ПИСЬМА
20 DIM N$(20),A$(20,3)
30 REM
40   GOSUB 120
50     GOSUB 160
60     FOR I=1 TO N
70       GOSUB 250
80     NEXT I
90     INPUT "ДРУГИЕ АДРЕСАТЫ (ДА, НЕТ)";B$
100 IF B$="ДА" THEN 50
110 STOP
120 REM ПОДПРОГРАММА ДЛЯ ВВОДА ПОСТОЯННОЙ ЧАСТИ ПИСЬМА
130   INPUT "ВВЕДИТЕ ДАТУ";D$
```

```
140 REM ЗДЕСЬ МОЖНО ВВЕСТИ И ДРУГИЕ ПОДОБНЫЕ ДАННЫЕ
150 RETURN
160 REM ПОДПРОГРАММА ДЛЯ ВВОДА ФАМИЛИЙ АДРЕСАТОВ
170   INPUT "ЧИСЛО ФАМИЛИЙ";N
180   FOR I=1 TO N
190     INPUT N$(I)
200     FOR J=1 TO 3
210        INPUT A$(I,J)
220     NEXT J
230   NEXT I
240 RETURN
250 REM ПОДПРОГРАММА ДЛЯ ПЕЧАТИ ПИСЬМА
260       RESTORE
270       K=4
280       L=30
290       GOSUB 440
300       PRINT TAB(35);D$
310       PRINT N$(I)
320       FOR M=1 TO 3
330     PRINT A$(I,M)
340       NEXT M
350       PRINT "ДОРОГОЙ ";N$(I);"!"
360       K=10
370       L=1
380       GOSUB 440
390       PRINT
400       PRINT TAB(35);"ИСКРЕННЕ ВАШ"
410   K=10
420   GOSUB 500
430 RETURN
440 REM ПОДПРОГРАММА ДЛЯ ЧТЕНИЯ/ПЕЧАТИ
450   FOR P=1 TO K
460     READ B$
470     PRINT TAB(L);B$
480   NEXT P
490 RETURN
500 REM ПОДПРОГРАММА ДЛЯ ПРОПУСКА ЗАДАННОГО ЧИСЛА СТРОК
510       FOR P=1 TO K
520          PRINT
530       NEXT P
540   RETURN
550 DATA        ------|
                      > Ваша фамилия и адрес (4 СТРОКИ)
                      |
                ------|
315
```

5.3 Когда каждый элемент данных (в данном случае сведений о человеке) составлен из нескольких значений, для упорядочения списка элементов бывает проще производить перестановки в списке указателей, а сами элементы оставлять на месте.

Уровень 1

Считать данные в два массива: фамилии в A$ ( ) , а номера — в B ( ).

Установить начальные значения указателей (элементов индексного массива I ( )) на запомненные элементы данных.

Отсортировать значения из B ( ), на которые указывает массив I ( ); при сортировке переставляются значения в I( ), а не в A$( ) и в B( ).

Распечатать список элементов данных в порядке, указанном в I (). Кроме считывания значений, остальные три действия (установка значений в индексном файле, сортировка, распечатка) детально описаны в подразд. 4.5.3. Приведенные в нем фрагменты можно использовать в качестве подпрограмм, если изменить в них некоторые имена переменных. В данном примере разработки по методу "сверху вниз" очень хорошо видно, чем можно воспользоваться на нижнем уровне функций/подпрограмм. Окончательная программа такова:

```
10 REM СОРТИРОВКА С ИСПОЛЬЗОВАНИЕМ ИНДЕКСА
20 DIM A$(10),B(10), I(10)
30 FOR K=1 TO 10
40   READ AS(K) ,B(K)
50 NEXT К
60 REM СОРТИРОВКА
70   GOSUB 900
80 REM ПЕЧАТЬ
90   GOSUB 120
100 STOP
110 REM
120 REM ПОДПРОГРАММА ПЕЧАТИ
130 FOR K=1 TO 10
140   PRINT A$(I(K));B(I(K))
150 NEXT К
900 REM ПОДПРОГРАММА СОРТИРОВКИ (С ИСПОЛЬЗОВАНИЕМ ИНДЕКСА)
910                 ----------1
> Программа из подразд. 4.5.3 с приведенными I ниже изменениями
1120 RETURN
316



1130 DATA
1140 END
```

Изменения к программе из подразд. 4.5.3:

```
910 FOR L=1 TO 10
1040 FOR I=10 TO 2 STEP -1
```

```
1060 IF B(I1)<=B(I2) THEN 1100
```
РЕШЕНИЯ УПРАЖНЕНИЙ К ГЛ. 7

7.1

```
10   DIM A(4,4),B(4,4),C(4,4),D(4,3),Z(4,4)
20   REM
30   PRINT "ВВЕДИТЕ ТРИ МАТРИЦЫ A, B, C"
40   MAT INPUT A,B,C
50   MAT A=A+B
60   MAT Z=A+C
70  MAT PRINT Z;
80  REM
90   PRINT "ВВЕДИТЕ МАТРИЦУ D"
100   MAT INPUT D
110   MAT A=Z*D
120   MAT PRINT A;
130   END
10 DIM A(10,10),B(10,10)
20 INPUT N
30 MAT A=CON(N,N)
40 MAT A=(5.)*A
50 REM
60 MAT B=IDN(N,N)
70 MAT B=(5.)*B
80 REM
90 MAT A=A+B
100 PRINT "ИСХОДНАЯ МАТРИЦА:"
110 MAT PRINT A;
120 REM
130 MAT B-INV(A)
140 PRINT "ОБРАТНАЯ МАТРИЦА:"
150 MAT PRINT B;
160 END
10 DIM A(4.4),B(4,4)
20 REM
30 MAT READ A
40 MAT B=A*A
50 PRINT "КВАДРАТ РАВЕН:"
60 MAT PRINT B;
```


```
70   REM
80   MAT B=B*A
90   PRINT "КУБ РАВЕН:"
100   MAT PRINT B;
110   REM
120   MAT B=B*A
130   PRINT "ЧЕТВЕРТАЯ СТЕПЕНЬ РАВНА:"
140   MAT PRINT B;
150   REM
```

```
160   REM ПРЕДПОЛАГАЕТСЯ, ЧТО ЧТЕНИЕ ИДЕТ ПО СТРОКАМ
170   DATA 1,2,-1,1,2,0,1,2,1,0,1
160   DATA 3,2,3,1,1
190   END
```

7.4

```
10 DIM A(3,3),B(3,3),C(3,3)
20 REM
30 MAT READ A
40 MAT B=A*A
50 MAT C=B*A
60 MAT B=(2.)*B
70 MAT C=C-B
60 MAT A=(9.)*A
90 MAT C=C-A
100 PRINT "МАТРИЦА A**3-2A**2-9A РАВНА:"
110 MAT PRINT C;
120 REM
130 DATA 2,1,3,1,-1,2,1,2,1
140 END
```

7.5 Как сами решения, так и расхождения в них при применении выбора главного элемента и без него очень сильно зависят от точности представления чисел в используемой ЭВМ. Невязки приведенного ниже решения имеют порядок 10^{-8}:

x_1 =5.003106553 x_2 =-5.072538054 x_3 =0.98897173

7.6 Программа GAUSS в процессе исключения переменных выполнит две перестановки уравнений (третьего с первым, третьего со вторым) и даст результат

```
2       1       1
0       1     -1 0       0-6
```

Таким образом, определитель равен $(-1)^2$ *2 *1 * (-6) =-12.

ПРИЛОЖЕНИЕ II. КОДЫ ASCII

По определению коды ASCII 7-битовые, поэтому они занимают диапазон 0 ... 127. В табл. П2.1 показано соответствие между кодами и символами:

Коды символов ASCII

Таблица П2.1

Десятичное значение	Символ	Десятичное значение	Символ	Десятичное значение	Символ
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	\
007	BEL	050	2	093	]
008	BS	051	3	094	↑ ИЛИ^
009	HT	052	4	095	←или _

010	LF	053	5	096	
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058		101	e
016	DLE	059	*	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	1
023	ETB	066	B	109	m
024	CANCEL	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	Пробел	075	К	118	**V**
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	|
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DELETE
042	*	085	U		

Обратите внимание на следующие аббревиатуры:
LF - Line feed (переход на новую строку) FF - Form feed (переход к новой странице) CR - Carriage return (возврат каретки)
Приведенные выше коды и коды
ESCAPE (отключить) CANCEL (отменить) DELETE (удалить)
широко употребляются. Некоторые другие коды имеют следующие значения:

SOH - Start of heading	(начало заголовка)
STX - Start of text	(начало текста)
ETX - End of text	(конец текста)
EOT - End of transmission	(конец передачи)
ENQ - Enquiry	(запрос)
ACK - Acknowledge	(подтверждение)

BEL - Bell	(звуковой сигнал)
BS - Backspace	(возврат на одну позицию)
HT — Horizontal tabulation	(горизонтальная табуляция)
VT - Vertical tabulation	(вертикальная табуляция)
SO - Shift out	(нижний регистр)
SI - Shift in	(верхний регистр)
DLE - Data link escape	(отключение линии передачи данных)
DC1 - DC4	(управление включением/выключением устройств)
NAK - Negative acknowledge	(налицо ошибочная ситуация)
SYN — Synchronous idle	(синхронизирующий заполнитель)

Во многих системах для кода выделяется байт (8 бит) и дополнительные значения кодов от 128 до 255 используются для специальных графических символов и символов управления курсором. В некоторых системах приведенные выше управляющие коды (с десятичными значениями от 0 до 31) используются в иных целях по сравнению со стандартом. Для уточнения деталей обратитесь к руководству по Вашей ЭВМ. На клавиатурах многих микроЭВМ нажатие клавиши A при предварительно нажатой и удерживаемой клавише CONTROL генерирует код ASCII, равный 001, CONTROL B дает код 002, и так далее до CONTROL Z, дающего код 026. Таким способом многие из управляющих символов ASCII могут быть при необходимости без труда набраны на клавиатуре.

ПРИЛОЖЕНИЕ III. КОМАНДЫ БЕЙСИКА

Ниже приведен перечень наиболее общеупотребительных команд Бейсика. Учтите, что многие операторы Бейсика могут быть использованы в режиме немедленного исполнения (см. разд. 2.4) и тем самым служить командами.

Детали формата команд меняются от системы к системе, причем ни у одной системы нет всех этих команд.

AUTO	Обеспечивает автоматическую нумерацию (обычно с приращением 10) очередной строки при наборе программы на Бейсике.
BYE	В мультитерминальных вычислительных системах (обычно не на микроЭВМ) эта команда завершает сеанс работы.
CLEAR или CLR	Обнуляет все используемые в программе переменные.

CLS	Гасит экран (или только изображение текста).
CLG	Гасит графическую часть изображения на экране.
CONT	Продолжает выполнение программы после ее приостанова. Значения переменных сохраняются и могут быть использованы при возобновлении исполнения программы.
CONTINUE 500	Расширенный вариант команды CONT, позволяющий возобновить исполнение программы с заданной строки (реализован только на больших ЭВМ).
DEL или DELETE	Удаляет всю текущую программу.

DELETE 10,20 DELETE 10-20	Удаляет из текущей программы строки с номерами от 10 до 20 (включительно).
EDIT 20	Редактирует строку 20 текущей программы (только в Бейсике Microsoft).
GET PROG	Используется на больших ЭВМ для загрузки программы из архивной памяти в оперативную.
HELLO	Используется в мультитерминальных системах для идентификации пользователя и начала сеанса работы.
HELP	Обращение за справкой по поводу команд или операторов Бейсика.
KILL DATA	Удаляет поименованный файл из архивной памяти (для больших ЭВМ).
KILL "DATA"	Версия приведенной выше команды в Бейсике Microsoft.
LIST	Изображает текущую программу на ВТУ.
LIST 20	Изображает только строку с номером 20, а в некоторых системах — и все последующие строки.
LIST 20,40 LIST 20-40	Изображает строки, номера которых находятся в указанном промежутке.
LIST, 100 LIST -100	Изображает программу от начала до строки с номером 100 (включительно).
LLIST	Аналогична команде LIST, но выводит текст программы на принтер вместо ВТУ.
LOAD	Загружает следующую программу с кассеты (в Бейсике Microsoft для этих целей используется CLOAD) .
LOAD "PROG"	Загружает поименованную программу с кассеты или диска. Для идентификации устройства может задаваться дополнительный параметр.
NAME PROG2	Присваивает текущей программе новое имя (для больших ЭВМ).
NAME PROG1 AS PROG2	Присваивает находящейся на диске программе новое имя (только в Бейсике Microsoft).
NEW	Полностью удаляет текущую программу. Кроме того, на микро-ЭВМ перезапускает интерпретатор и приводит его в начальное состояние.
NEW PROG	Удаляет текущую программу и дает имя новой программе (только для больших ЭВМ).
OLD	Отменяет действие NEW при условии, что не было набрано ни одной новой строки (только в Бейсике BBC).

OLD PROG	То же, что GET для больших ЭВМ.
RENUMBER RENUM RESEQUENCE	Перенумерация строк текущей программы, обычно с приращением 10, но возможны дополнительные параметры для указания

	другого приращения.
RUN	Начинает исполнение текущей программы с самого начала. В отличие от CONT, перед запуском обычно обнуляет все переменные.
RUN 500	То же, что и RUN, но начинает исполнение программы со строки 500.
SAVE	Сохраняет текущую программу на магнитофонной кассете (версия Microsoft использует CSAVE).
SAVE	Используется в больших ЭВМ для сохранения текущей программы в архивной памяти при условии, что ранее ей было дано имя командой NAME.
SAVE "PROG"	Сохраняет текущую программу на диске или магнитофонной кассете.
SCRATCH или SCR	То же, что DELETE.
SYSTEM	Выход из Бейсика и возврат управления операционной системе CP/M (только в Бейсике Microsoft).
UNSAVE PROG	То же, что и KILL PROG.

ПРИЛОЖЕНИЕ IV. СЛОВАРЬ ТЕРМИНОВ

Ниже разъясняются некоторые специальные термины, упоминающиеся в данной книге; во многих случаях полное толкование уже приводилось в тексте.

Адрес	Код, идентифицирующий ячейку памяти. Служит для указания ячейки при чтении или записи данных
ANSI	Американский национальный институт стандартов
Арифметико-логическое устройство	Часть аппаратных средств ЭВМ, выполняющая различные арифметические и логические операции
Архивная память	Память для долговременного хранения программ и данных. Обычно представляет собой магнитный носитель: жесткий или гибкий диск, кассету или бобину с магнитной лентой
Ассемблер	Программа, преобразующая текст на языке ассемблера в машинные коды. Она гораздо меньше компиляторов, предназначенных для работы с языками программирования высокого уровня
Байт	Основная совокупность битов, которую ЭВМ обрабатывает как единое целое; ее длина составляет 8 бит
Байт состояния	Байт, значение которого устанавливается аппаратурой ЭВМ для описания текущего состояния операции по вводу-выводу данных, например готово ли

	устройство к новому обмену данными
Безопасная печать	Способ печатания потенциально важных документов, например чеков

Бит	Двоичная цифра, которая может принимать только значения 0 и 1
Большая ЭВМ	Мощная ЭВМ, физически большая и способная обслуживать много терминалов одновременно
Вектор	Простой (одномерный) массив; список значений, для которых существенна занимаемая позиция
Вложение	Включение одного элемента внутрь другого; обычно этот термин применяется к циклам FOR-NEXT языка Бейсик для указания на то, что один из них полностью содержится в другом
Выравнивание влево	Выравнивание элементов в различных зонах или строках по левому краю зоны или строки
Графика с низким разрешением	Генерация линий, кривых и цветовых пятен с помощью символов и знаков, а не отдельных мелких точек
Диалоговый режим	Непосредственно общение пользователя с ЭВМ
Дисковый контроллер	Устройство для перемещения головок чтения/записи, а также управления процессом чтения данных с диска и записи на него
ЗУПВ (запоминающее устройство с произвольной выборкой)	Память, в обычных условиях доступная как для чтения, так и для записи
Интерпретатор	Программа, обрабатывающая один за одним операторы программы на языке высокого уровня и исполняющая каждый оператор по мере его обработки. В отличие от компилятора, не создает программу на языке низкого уровня
Искусственный интеллект	Желанная, но пока недостижимая способность прибора к выполнению функций, обычно присущих человеческому разуму
Ключ записи	Часть записи, используемая для ее идентификации. Например, фамилия может быть ключом записи, состоящей из фамилии и адреса

Компилятор	Программа, преобразующая программу на языке высокого уровня (например, на Бейсике) в программу на языке низкого уровня (на языке ассемблера или в машинных кодах)
Курсор	Метка на экране ВТУ, показывающая место появления очередного символа, набираемого на клавиатуре
Логические выражения	Выражения, составленные с помощью операций сравнения значений и операций И, ИЛИ и исключающее ИЛИ и принимающие значения ИСТИНА или ЛОЖЬ
Логический номер	Логический адрес, служащий для привязки к устройству ввода-вывода ЭВМ. Действительный машинный адрес устройства определяется по его логическому адресу системой с Бейсиком
Локальные переменные	Переменные, имеющие значение (существующие) в пределах ограниченного фрагмента программы и

Матрица	никак не связанные с одноименными переменными используемыми вне этого фрагмента. Например, в Бейсике некоторые блочные функции могут иметь локальные переменные Двумерный массив; таблица значений, в которой существенна позиция значения
Модули	Частично отделенные от программы блоки операторов, например подпрограммы или функции
Начальный загрузчик	Средство для загрузки первых команд в пустую память ЭВМ после включения питания. Обычно эта очень простая группа команд загружает в память остальную часть программы начального вызова, которая уже может быть использована для загрузки любой другой программы, в том числе и операционной системы. Начальный загрузчик чаще всего представляет собой специальное ПЗУ
Немедленное исполнение (в Бейсике)	Исполнение оператора или команды сразу же после завершения ввода в результате нажатия на клавишу возврата каретки или на клавишу SEND
Обратная матрица (только для	Матрица, при умножении которой на исходную матрицу получается единичная

квадратной матрицы)	матрица
Он-лайн	Относится к терминалу и означает, что терминал подключен к ЭВМ и обслуживается ею. Эта ситуация всегда имеет место при работе с микроЭВМ, рассчитанной на обслуживание одного пользователя
Определитель (только для квадратной матрицы)	Числовое значение, получаемое как сумма произведений элементов матрицы
Память	Область с непосредственным доступом, в которой хранятся программы и данные во время обработки
ПЗУ (постоянное запоминающее устройство)	Содержит данные, занесенные в него в процессе изготовления и не требует электропитания для их сохранения
Пошаговая детализация	Последовательное создание более детальных (более низких) уровней разработки при применении структурного программирования
Преобладающая диагональ	Главная диагональ матрицы, обладающая тем свойством, что абсолютное значение каждого ее элемента превышает сумму абсолютных значений остальных элементов строки, в которой этот элемент находится
Программа	Последовательность команд или операторов, задающая выполнение определенных действий. В Бейсике все операторы программы воспринимаются друг за другом в порядке возрастания их номеров
Присоединенная матрица (только для квадратной матрицы)	Квадратная матрица, элементы которой являются алгебраическими дополнениями элементов исходной матрицы. (Алгебраическим дополнением называется взятый с определенным знаком определитель матрицы, полученной путем удаления одного столбца и одной строки из исходной матрицы)

произвольный доступ	Возможность доступа ко всем элементам данных примерно за одно и то же время. Не требует чтения или записи соседних элементов
Реакция системы	Промежуток времени между завершением набора команды и появлением результатов на ВТУ или принтере

Редактор	Программа, манипулирующая всеми видами текстов программ (и, может быть, файлами данных) и позволяющая пользователю вносить в них изменения, добавлять новые данные и удалять старые
Рекурсия	Этот термин применяется к подпрограмме или блочной функции и означает, что во время исполнения она вызывает самое себя
-Синтаксис	Правила написания операторов языка, т. е. что и в каком месте оператора может появиться
Стандарт минимального подмножества Бейсика (Minimal BASIC)	Документ Европейской ассоциации производителей вычислительной техники, определяющий основное ядро языка Бейсик. Этот стандарт, ECMA-55, был опубликован в 1977 году и может быть бесплатно получен по следующему адресу: ECMA, 114 Rue du Rhone, 1204 Geneva, Switzerland
Стек	Область памяти ЭВМ, используемая по правилу "первым пришел, последним ушел". В верхней части стека запоминается последний элемент данных, который удаляется первым
Структурное проектирование	Специальный подход к разработке программ, при котором работа идет от описания общих к описанию более детальных действий
Таблицы и списки компилятора	В процессе компиляции образуется множество вспомогательных данных в виде таблиц и списков, для хранения которых требуется рабочая область памяти
Путь исполнения	Путь следования управления при исполнении программы на языке Бейсик, т. е. последовательный список операторов, которые должны исполняться после набора команды RUN
Численная устойчивость	В устойчивой численной задаче небольшие изменения исходных данных не приводят к кардинальному изменению ее решения
Шестнадцатеричная система	Система счисления с основанием 16, в которой допустимыми цифрами считаются десятичные цифры от 0 до 9 и буквы от A до F
Шина данных IEEE 488	Группа из 24 параллельных проводников (или контактных гнезд) с определенными свойствами. Служит для обмена данными между приборами и ЭВМ
Язык ассемблера	Язык программирования, использующий более или менее удобочитаемую мнемонику для машинных команд, понимаемых непосредственно ЭВМ. Обычно каждой

машинной команде соответствует одна
команда на языке ассемблера

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

СОДЕРЖАНИЕ