В.Э.БАЛАСАНЯН
С.В.БОГДЮКЕВИЧ
В.А.ШАХВЕРДОВ

ПРОГРАММИРОВАНИЕ НА МИКРОЭВМ "ИСКРА 226"

МОСКВА
"ФИНАНСЫ И СТАТИСТИКА" 1987

ББК 32.973—01
Б20
УДК 681.3.06

Р е ц е н з е н т ы :

д-р техн. наук Э. А. ТРАХТЕНГЕРЦ и канд. экон. наук А. А. АЗЕЕВ

Баласанян В. Э. и др.

Б20 Программирование на микроЭВМ «Искра 226» / В. Э. Баласанян, С. В. Богдюкевич, В. А. Шахвердов. — М.: Финансы и статистика, 1987. — 264 с: ил.

Описываются средства и способы программирования на новой отечественной микроЭВМ «Искра 226», предназначенной для эксплуатации непосредственно на рабочих местах в органах управления различного уровня. Рассматривается широко распространенный диалоговый язык программирования Бейсик. Изложение сопровождается примерами.

Для специалистов различных отраслей народного хозяйства, использующих микроЭВМ «Искра 226» в своей практической деятельности,

2405000000—001
Б————————ББК 32.973—01
010(01)—87

ОГЛАВЛЕНИЕ

В настоящее время специалисты самых различных областей получили возможность использовать ЭВМ в своей работе для решения стоящих перед ними задач.

Одним из самых распространенных языков программирования среди все увеличивающейся группы пользователей-непрофессионалов ЭВМ, и в первую очередь среди пользователей персональных компьютеров, является Бейсик. Популярность Бейсика объясняется простотой его синтаксиса, ориентацией на диалоговый характер программирования.

В предлагаемой читателю книге описано программирование на весьма мощной версии Бейсика, реализованной на отечественной микроЭВМ «Искра 226», которая является настольной ЭВМ индивидуального применения. Взаимодействие пользователя с машиной осуществляется посредством интерпретирующей Бейсик-системы, которая обеспечивает выполнение всех необходимых функций по управлению работой машины и программированию на ней. При этом имеющиеся в Бейсике средства практически исключают необходимость программирования на ассемблере или в машинных командах.

Чтение книги не требует специальной подготовки в области вычислительной техники и программирования. Термины и понятия разъясняются по мере появления в тексте, а формальная символика сводится к минимуму. Основное внимание уделяется содержательному описанию семантики операторов, особенностям их использования и техническим приемам программирования. В книге приводится большое количество примеров программ и их типовых фрагментов, которые могут быть непосредственно использованы при программировании разнообразных практических задач.

Часть I содержит вводные сведения о микроЭВМ «Искра 226», необходимые для работы на машине.

В части II излагаются основы программирования на базовом подмножестве Бейсика «Искры 226», а также сведения по вводу. Редактированию, отладке и каталогизации программ. Материал строится таким образом, чтобы читатель с самого начала мог приступить к написанию и реализации на ЭВМ программ — сначала элементарных, а затем и более сложных.

Часть I ОСНОВНЫЕ СВЕДЕНИЯ О МИКРОЭВМ „ИСКРА 226“ И РАБОТЕ НА НЕЙ

Глава 1. МИКРОЭВМ «ИСКРА 226»

1.1. Состав устройств машины

МикроЭВМ «Искра 226» является машиной индивидуального (персонального) пользования, т. е. она рассчитана на одновременную работу на ней одного человека. Конструктивно «Искра 226» выполнена в виде настольного центрального блока, содержащего ряд основных устройств машины, и подключаемых к нему дополнительных внешних устройств. Количество и состав внешних устройств могут варьироваться для различных моделей (исполнений) машины. По требованиям к электропитанию и окружающей среде микроЭВМ «Искра» не отличается от обычных электробытовых устройств.

Центральный блок микроЭВМ «Искра 226» включает следующие устройства:

1. Процессор, выполненный на микропроцессорных интегральных схемах, который обеспечивает выполнение вычислительных операций (в заводской документации для простоты изложения процессором называют весь центральный блок машины).

2. Запоминающее устройство, выполненное на интегральных схемах, для оперативного хранения программ и данных. В него входят постоянное запоминающее устройство (ПЗУ) емкостью 16 Кбайт, управляющая память (УП) емкостью 64 Кбайт, оперативное запоминающее устройство (ОЗУ) емкостью 64 Кбайт[1].

В ПЗУ хранятся системные программы — загрузчик и транслятор с ассемблера, записанные в него при изготовлении машины. В отличие от ПЗУ содержимое УП и ОЗУ требуется перезаписывать после каждого включения машины. В УП при подготовке машины к работе записывается (или, как часто говорят, загружается) с магнитного диска программа — интерпретатор языка Бейсик. ОЗУ используется для размещения программ на Бейсике и данных, обрабатываемых этими программами.

3. Дисплей с выносной клавиатурой, встроенный в переднюю часть центрального блока и являющийся основным устройством вывода информации. На него может выводиться как символьная информация (т. е. алфавитно-цифровые символы фиксированного размера в фиксированных позициях экрана), так и графическая информация в виде точек, с помощью которых можно синтезировать произвольные изображения[2].

Выносная клавиатура служит для ввода информации в машину. Помимо клавиш, соответствующих определенным алфавитно-цифровым символам, она имеет клавиши для ввода целых слов Бейсика и специальных команд управления, а также свободные клавиши, функции которых могут задаваться пользователем.

4. Устройства сопряжения для подключения внешних устройств, называемые в заводской документации блоками интерфейсными функциональными (БИФ). Эти блоки примыкают к задней стенке процессора, и к их разъемам непосредственно подключаются соответствующие внешние устройства.

Отдельно следует сказать о телекоммуникационном интерфейсе, состоящем из микроЭВМ и программируемого узла связи с аппаратурой передачи данных. Он обеспечивает автономные прием-передачу и предварительную обработку информации в режиме телеобработки.

В число внешних (периферийных) устройств машины входят накопители на магнитных дисках и лентах, печатающее устройство, планшетный графопостроитель.

Для длительного хранения программ и данных в микроЭВМ «Искра 226» используются магнитные диски. В состав машины могут входить два типа накопителей на магнитных дисках: накопители на гибких магнитных дисках и накопители на жестких (кассетных) магнитных дисках.

Гибкий магнитный диск (дискета) представляет собой тонкий пластмассовый диск размером с гибкую грампластинку, на который нанесен магнитный слой. Для предохранения от повреждений дискета запрессована в пластмассовый конверт, имеющий прорезь для головки записи—чтения. В накопитель могут быть одновременно вставлены две дискеты. Запись информации на гибкий магнитный диск осуществляется в формате ЕС ЭВМ, что позволяет использовать диски для обмена

[1] Байт — единица емкости запоминающих устройств; одного байта достаточно для хранения одного алфавитно-цифрового символа; 1 Килобайт = 1024 байтам, 1 Мегабайт =1024 Кбайтам.

[2] В заводской документации дисплей называется блоком отображения символьно-графической информации (БОСГИ).

данными между «Искрой 226» и машинами ЕС.

Жесткий магнитный диск представляет собой металлический диск размером с обычную грампластинку. Емкость диска составляет около 2,5 Мбайт. В накопителе имеются один встроенный диск (так называемый фиксированный) и один съемный. Съемные диски заключены в защитную пластмассовую кассету.

Для архивного хранения программ и данных, а также для обмена информацией с другими ЭВМ может использоваться накопитель на девятидорожечной магнитной ленте емкостью да 10 Мбайт. Данные на магнитную ленту записываются в формате ЕС ЭВМ.

Печатающее устройство позволяет выводить данные и программы на бумажные носители. В зависимости от типа устройства скорость печати может колебаться от 40 до 180 символов в секунду на перфорированную бумагу или отдельные листы. Для вывода графических данных используется планшетный графопостроитель.

Кроме перечисленных устройств в состав отдельных исполнений машины входят интерфейсы для сопряжения микроЭВМ «Искра 226» с внешними устройствами СМ ЭВМ, а также устройства цифро-аналогового и аналого-цифрового преобразования для сопряжения с контрольно-измерительной аппаратурой.

Следует отметить, что большой набор внешних устройств в сочетании с возможностью установки машины непосредственно на рабочем месте обеспечивает самую широкую сферу применения микроЭВМ «Искра 226». Она может использоваться как автономно, так и в качестве интеллектуального терминального устройства сети ЭВМ. На «Искре 226» можно выполнять разнообразные расчеты, подготовку текстовых документов, создавать информационно-поисковые системы, осуществлять автоматизированное проектирование и управление научными экспериментами и технологическими процессами и т. д.

1.2. Программное обеспечение

В состав системного программного обеспечения микроЭВМ «Искра 226» входят интерпретатор языка Бейсик, а также загрузчик интерпретатора и транслятор с ассемблера.

Написанная на Бейсике программа представляет собой последовательность операторов — команд, состоящих из ключевых слов и символов Бейсика и предписывающих выполнение определенных операций. Интерпретатор осуществляет пошаговый перевод (трансляцию) операторов языка Бейсик в машинные команды и немедленное их исполнение. Программы пользователя хранятся в исходной форме, т. е. на Бейсике, что существенно упрощает их отладку и модификацию.

Загрузчик осуществляет загрузку интерпретатора с магнитного диска в управляющую память, а при необходимости и запись интерпретатора на магнитный диск. Он обеспечивает также запуск тестовых программ, позволяющих проверить работоспособность процессора, оперативной и управляющей памяти, дисплея, клавиатуры и накопителей на магнитных дисках, а кроме того, выполняет ряд других функций по управлению работой машины.

Наличие транслятора с ассемблера дает возможность включать в Бейсик-программы фрагменты, написанные на машинно-ориентированном языке. Хотя практически все необходимые функциональные возможности машины могут быть реализованы с помощью Бейсика, в особых случаях использование ассемблерных фрагментов может оказаться полезным, например для обеспечения большего быстродействия.

Глава 2. ОСНОВЫ РАБОТЫ НА МИКРОЭВМ «ИСКРА 226»

2.1. Особенности дисплея и клавиатуры

Дисплей и клавиатура, являющиеся основными устройствами, обеспечивают взаимодействие пользователя с микроЭВМ «Искра 226».

Дисплей (рис. 2.1), реализованный на электроннолучевой трубке, служит для вывода символьной и графической информации: текста, набираемого пользователем на клавиатуре, а также результатов выполнения операций[1].

[1] Графический режим в данной книге не рассматривается.

Символьная информация выводится в виде алфавитно-цифровых символов. Набор высвечиваемых символов включает заглавные и прописные буквы русского и латинского алфавитов, цифры и служебные знаки (см. приложение 1). Символы высвечиваются светлым контуром на темном фоне (негативное изображение) или темным контуром на светлом фоне (позитивное изображение) в фиксированных позициях экрана (24 строки по 80 символов в каждой). При выводе на экран символы заполняют его по строкам: после заполнения первой строки начинает заполняться вторая и т. д. до тех пор, пока не будет заполнена последняя 24-я строка. Затем содержимое экрана сдвигается на одну строку вверх.

Рис. 2.1. Процессор микроЭВМ «Искра 226»

Дисплей обеспечивает также генерацию звукового сигнала фиксированных тона и длительности.

На дисплее имеются регулятор контрастности свечения и кнопка системного сброса, используемая в аварийных ситуациях.

Клавиатура машины (рис. 2.2) позволяет вводить программы и данные, а также управлять работой машины. Для удобства пользования большая часть слов Бейсика может вводиться не только по буквам, но и целым словом одним нажатием клавиш.

В обычном режиме каждый набираемый на клавиатуре символ или слово Бейсика высвечивается на экране. При этом место, на котором высветится очередной символ, автоматически указывается с помощью курсора (подстрочной черты).

На клавиатуре можно выделить восемь функциональных зон, причем отдельные клавиши повторяются в различных зонах.

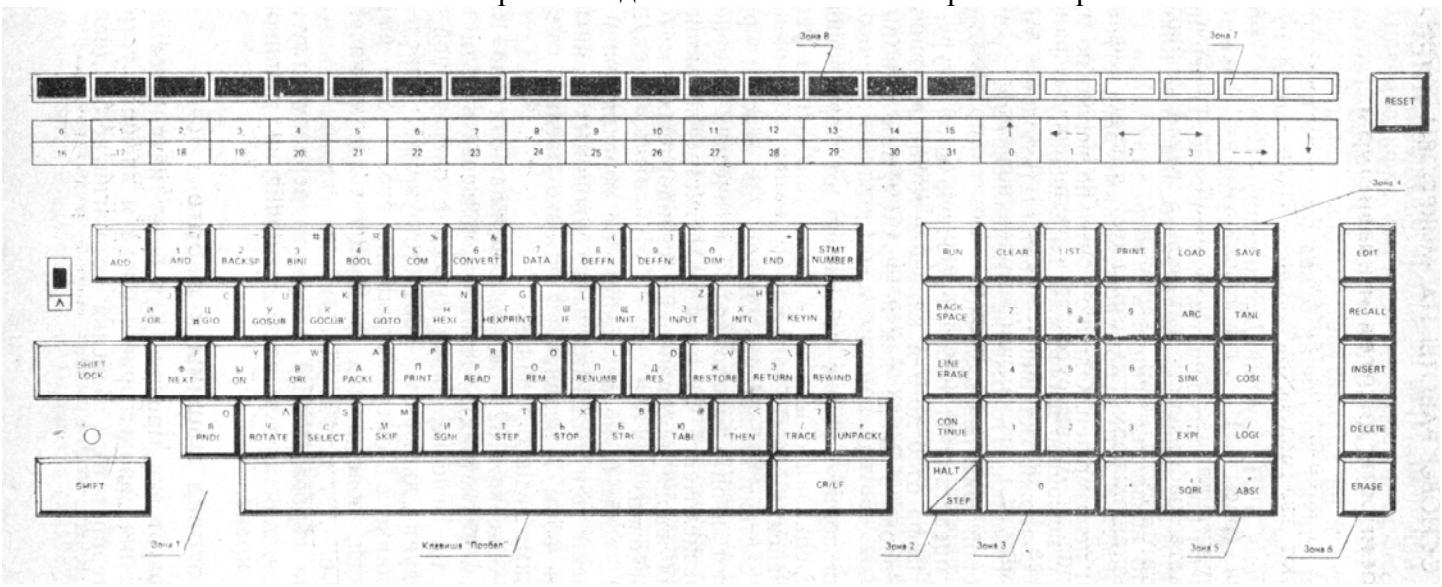

Рис.2.2. Клавиатура микроЭВМ «Искра 226»

Зона 1 соответствует обычной клавиатуре пишущей машинки и имеет два регистра – нижний и верхний. При нажатой клавише SHIFT включается верхний регистр, который может быть зафиксирован с помощью клавиши SHIFT LOCK (включение верхнего регистра индицируется лампочкой на клавиатуре). В нижнем регистре каждой клавише соответствуют заглазная буква русского или латинского алфавита, цифры и служебные знаки в зависимости от положения переключателя Р/Л (русский/латинский). Следует помнить, что, хотя некоторые русские и латинские буквы совпадают по начертанию, машиной они распознаются как разные. В верхнем регистре нажатием каждой клавиши вводится то или иное ключевое слово Бейсика, что позволяет ускорить набор этих слов и исключить ошибки. (Заметим, что в обычном режиме ввод строчных букв с клавиатуры невозможен. Однако довольно несложно организовать такой ввод с помощью программы на Бейсике.)

Зона 2 содержит клавиши, соответствующие управляющим клавишам Бейсика. Положение переключателя регистров для клавиш второй зоны безразлично.

Зона 3 представляет собой цифровую клавиатуру, состоящую из клавиш цифр от 0 до 9 и клавиши точки. Клавиши этой зоны позволяют вводить числа как в верхнем, так и в нижнем регистрах.

Клавиши зоны 4 также не зависят от переключения регистров, они предназначены для ввода часто используемых ключевых слов Бейсика.

Зона 5 состоит из восьми клавиш. В нижнем регистре они обеспечивают ввод символов арифметических операций, скобок и знаков-разделителей. В верхнем регистре клавиши этой зоны служат для ввода обозначений стандартных математических функций Бейсика.

Клавиши зоны 6, не зависящие от переключения регистров, предназначены для редактирования вводимых данных.

Клавиши зоны 7 клавиатуры также предназначены для работы в режиме редактирования.

Зона 8 содержит 32 «свободные» клавиши (по 16 в нижнем и в верхнем регистрах). Чтобы использовать эти клавиши, в программу на Бейсике, находящуюся в оперативной памяти машины, следует включить соответствующие определения. На съемной планке с номерами клавиш этой зоны имеется место, для надписей, поясняющих функциональное назначение клавиш в программе.

Клавиша сброса RESET в правом верхнем углу клавиатуры используется для прекращения выполнения машиной операций и очистки экрана.

2.2. Подготовка машины к работе

Этап подготовки машины к работе начинается с включения электропитания процессорного блока и внешних устройств и загрузки интерпретатора Бейсика с магнитного диска в управляющую память.

Рассмотрим последовательность действий при загрузке. После включения электропитания процессора на экране дисплея появляется сообщение

```
ЗАГРУЗЧИК
:_
```

которое свидетельствует, что включение прошло нормально и ЭВМ находится в режиме загрузчика. В этом режиме можно осуществлять ряд служебных операций, в том числе загрузку интерпретатора Бейсика с магнитного диска, запись загруженного интерпретатора на магнитный диск, запуск интерпретатора, а также ряд других функций по управлению работой машины.

Управление работой загрузчика происходит с помощью специальных команд, набираемых на клавиатуре. В случае ошибки набора для перемещения курсора влево и стирания последнего набранного символа можно пользоваться клавишей BACKSPACE. Набор команды завершается нажатием клавиши CR/LF, после чего начинается ее выполнение. Появление на экране двоеточия и курсора в начале следующей строки означает, что команда выполнена и машина готова к вводу очередной команды.

Для загрузки интерпретатора необходимо набрать команду загрузки — ключевое слово LOAD (нажать клавишу с обозначением этого слова или набрать его по буквам) и адрес дискового устройства, с которого будет осуществляться загрузка. Для обращения к правому или нижнему накопителю на гибких магнитных дисках (в зависимости от модели накопителя) в качестве адреса задается буква F, а при обращении к левому или нижнему — буква R. Если интерпретатор загружается с накопителя на жестком магнитном диске, то для обращения к фиксированному диску задается адрес F/1C, а при обращении к съемному диску — R/1C.

Так для загрузки интерпретатора с правого или нижнего накопителя на гибких магнитных дисках достаточно указать

```
:LOAD F_
```

а с левого или верхнего

```
:LOAD R_
```

и нажать клавишу возврата каретки CR/LF.

Если интерпретатор считывается с накопителя на жестких магнитных дисках, то для обращения к фиксированному диску следует указать

```
:LOAD F/1C, [CR/LF]
```

а при обращении к съемному диску

```
:LOAD R/1C, [CR/LF]
```

Примечание. Квадратные скобки означают здесь, что клавиша CR/LF на экране не высвечивается. В дальнейшем необходимость нажатия этой клавиши по окончании набора команд в большинстве случаев будет подразумеваться по умолчанию.

На диске могут быть записаны различные версии интерпретатора. По умолчанию осуществляется загрузка версии с номером 0. Для загрузки версии интерпретатора с определенным номером он набирается после адреса устройства в виде #NN, где NN это одна или две шестнадцатеричные цифры

(обозначаются цифрами от 0 до 9 и латинскими буквами от A до F). Допустим, для загрузки версии с номером 0 с гибкого диска, вставленного в левый (верхний) карман накопителя, необходимо набрать на клавиатуре

```
:LOAD R, #0 [CR/LF]
```

Для загрузки версии с номером 1 с гибкого диска, вставленного в правый (нижний) накопитель, нужно набрать на клавиатуре

```
:LOAD F,#1 [CR/LF]
```

Для загрузки с фиксированного жесткого диска версии с номером 21 нужно набрать на клавиатуре

```
:LOAD F/1C,#21 [CR/LF]
```

и т. д.

В случае успешного выполнения команды загрузки на экране появляется сообщение

```
BASIC 02 05.10.84
:_
```

в котором содержится наименование языка (Бейсик 02) и дата последней редакции загруженной версии интерпретатора на заводе-изготовителе.

В режиме загрузчика при необходимости можно записать на диск загруженный в машину интерпретатор. В команде записи SAVE адресация накопителя на дисках осуществляется так же, как и в команде загрузки. Например, для записи интерпретатора с номером версии 1 на гибкий диск R дисковода используется команда

```
:SAVE R#1 [CR/LF]
```

Для записи на фиксированный жесткий диск интерпретатора под номером 26 нужно выполнить команду

```
:SAVE F/1C#26 [CR/LF]
```

Номер, задаваемый при записи, определяет адрес на диске, начиная с которого должен быть записан интерпретатор (см. 2.3).

Для запуска интерпретатора, загруженного в управляющую память машины, необходимо набрать команду

```
:RUN 1 [CR/LF] 12
```

На экране дисплея появляется сообщение

```
READY BASIC 02 05.10.84
:_
```

Данное сообщение означает, что начал работать интерпретатор языка Бейсик, и с этого момента все взаимодействие с машиной осуществляется с помощью языковых средств Бейсика. На этом заканчивается этап подготовки машины к работе. Как правило, он занимает не более двух-трех минут с момента включения машины.

Возврат из Бейсик-системы в режим загрузчика может быть осуществлен нажатием кнопки системного сброса, находящейся на процессорном блоке. Системный сброс может понадобиться при сбоях или отказах машины, последствия которых, оставаясь в Бейсик-системе, устранить невозможно, при необходимости загрузить другую версию интерпретатора и т. д.

После нажатия кнопки системного сброса на экране появляется соответствующее сообщение и машина переходит в режим загрузчика

```
RESET
:_
```

Повторный запуск интерпретатора осуществляется, как и ранее, с помощью команды

```
:RUN 1 [CR//LF]
```

В этом случае все содержимое оперативной памяти, занесенное в нее при работе (текст программы на языке Бейсик, значения переменных и т. д.), стирается. При необходимости сохранения содержимого оперативной памяти следует пользоваться командой

```
:RUN 2 [CR/LF]
```

2.3. Форматирование магнитных дисков

Информация записывается на магнитный диск по концентрическим дорожкам, которые в свою очередь делятся на секторы. В каждый сектор записывается 256 байт информации. Секторы гибкого диска имеют порядковые номера с 0 по 1000, жесткого диска — с 0 по 9971. В процессе работы диск вращается с постоянной скоростью, а головка чтения-записи двигается по радиусу диска. Время обращения к сектору при этом складывается из времени радиального движения головки до дорожки с искомым сектором и ожидания прохождения сектора под головкой. Накопитель на магнитных дисках является запоминающим устройством прямого доступа, поскольку время обращения к сектору практически не зависит от его местонахождения.

Для подготовки к работе новых магнитных дисков необходимо выполнить так называемую процедуру форматирования, в ходе которой диск размечается на секторы. Форматирование диска осуществляется в режиме интерпретатора с помощью оператора Бейсика PRINT и шестнадцатеричной функции HEX. Для форматирования правого или нижнего гибких дисков следует набрать

```
:PRINT/18,  HEX(1B000000000400);
```

и нажать клавишу CR/LF. Для форматирования левого или верхнего гибкого диска

```
:PRINT/18,  HEX(1B000100000400);
```

Для фиксированного или съемного дисков следует набрать соответственно

```
:PRINT/1C,  HEX(1B000000000400);
```

или

```
:PRINT/1C,  HEX(1B000100000400);
```

После успешного выполнения форматирования диск может использоваться для записи и считывании информации.

Если в ходе эксплуатации диска обнаружены ошибки чтения-записи, целесообразно осуществить повторное форматирование. При этом следует иметь в виду, что в ходе форматирования диска ранее записанная на него информация стирается.

Заметим, что при записи интерпретатора на магнитный диск номер версии, задаваемый пользователем в команде записи, определяет номер сектора, начиная с которого будет записан интерпретатор. Интерпретатор занимает на диске 249 подряд идущих секторов. Версия с номером 0 записывается в секторах с 0 по 248, версия с номером 1—с 249 по 497 и т. д. Номер начального сектора S, с которого будет записана версия с номером *N,* может быть определен по следующей формуле:

$$S=N*249$$

Так, например, версия интерпретатора с шестнадцатеричным номером 26 будет записана начиная с сектора

$$S = 38*249 = 9462$$

где десятичное число 38 соответствует шестнадцатеричному числу 26[1]

$$26_{16}=2*16+6 = 38$$

2.4. Понятие о режиме непосредственного счета

После запуска интерпретатора машина переходит в так называемый *режим непосредственного счета.* В этом режиме машина немедленно исполняет команды пользователя. Режим непосредственного счета используется для управления работой машины, в том числе для подготовки программ и для выполнения несложных разовых вычислений. После запуска программы и до прекращения ее выполнения машина находится в программном режиме, в котором действия машины определяются последовательностью операторов выполняемой программы.

Взаимодействие с машиной в режиме непосредственного счета осуществляется в форме диалога. Наличие на экране двоеточия и курсора

```
:_
```

означает, что система готова к приему очередных команд пользователя. Чтобы выполнить один или

[1] О переводе чисел из одной системы счисления в другую см. подробнее в гл. 14.

несколько операторов Бейсика, пользователь набирает эти операторы на клавиатуре, разделяя их двоеточием, и нажимает клавишу окончания ввода CR/LF. Введенная последовательность операторов, называемая *строкой непосредственного счета,* анализируется машиной. Если операторы в строке набраны без ошибок, машина выполняет соответствующие набранным операторам операции и переходит к ожиданию ввода новой строки. Если же операторы набраны не по правилам синтаксиса Бейсика или при их выполнении возникли ошибки, строка не выполняется. На экран выводится сообщение об ошибке, и машина снова переходит к ожиданию ввода следующей строки.

В отличие от программ строки непосредственного счета не запоминаются машиной и, следовательно, не могут быть выполнены повторно (за исключением последней введенной строки[1]).

Рассмотрим работу в режиме непосредственного счета на примере использования оператора PRINT для выполнения элементарных арифметических вычислений, подобно тому как это делается в калькуляторах. Оператор PRINT выводит на экран результат вычисления арифметического выражения, заданного после ключевого слова PRINT. Например, необходимо вычислить значение следующего арифметического выражения

20/(2+2/\3) +24*5/4—1.2

Примечание. В Бейсике для обозначения деления используется символ /, умножения — *, возведения в степень — /\, для отделения десятичной дробной части числа — точка.

Наберем сначала с клавиатуры ключевое слово PRINT (по буквам или одним нажатием клавиши со словом PRINT, расположенной в зоне 4 клавиатуры). На экране дисплея появится

:PR1NT__

Курсор показывает позицию, на которой появится следующий набираемый символ. Завершим набор

:PRINT 20/(2 + 2/\3)+24*5/4—1.2__

Пробелы в строке интерпретатором игнорируются. В случае возникновения ошибок можно пользоваться клавишами BACKSPACE или LINE ERASE зоны 2 клавиатуры. При нажатии клавиши BACKSPACE последний набранный символ стирается и курсор перемещается на его место. При нажатии клавиши LINE ERASE стирается вся набранная строка непосредственного счета.

После нажатия клавиши CR/LF машина выведет на экран результат вычислений

```
:PRINT 20/(2 + 2/\3)+24*5/4—1.2 [CR/LF]
30.8
:__
```

При выполнении вычислений можно использовать математические функции, включенные в Бейсик. Например, для вычисления натурального логарифма от значения выражения

8/2+1

достаточно набрать оператор

: PRINT LOG (8/2+1) _

и нажать клавишу CR/LF.

```
1.609437912434
:__
```

Для вычисления значений нескольких выражений можно включать в строку непосредственного счета несколько операторов PRINT, разделяя их двоеточием. Например,

```
:PRINT 2/\2:PRINT 2/\3:PRINT 2/\4  [CR/LF]
4
8
16
:__
```

Если во введенной строке содержалась ошибка, то вместо выполнения оператора будут выданы сообщение об ошибке и ее код. Например,

```
:PRINT   11/2+130+   [CR/LF]
                    /\ERR  06
```

[1] Подробно об этом говорится в 3.3.

:_

Стрелка в этом сообщении указывает на место ошибки во введенной строке, а 06 обозначает код синтаксической ошибки при вводе с клавиатуры. В этом случае для выполнения вычисления пользователь должен повторить оператор, набрав его правильно[1].

Заметим в заключение, что длина строки непосредственного счета не должна превышать 253 символа, включая двоеточия, используемые для разделения операторов, и пробелы. Так как длина строки экрана равна 80 символам, то строка непосредственного счета может размещаться в пределах четырех экранных строк.

[1] Более подробные сведения об использовании оператора PRINT для вычисления значений арифметических выражений содержатся в гл. 4.

Часть II ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ БЕЙСИК

Глава 3. ЭЛЕМЕНТЫ ЯЗЫКА БЕЙСИК

3.1. Что такое программа на языке Бейсик

Язык Бейсик относится к так называемым языкам программирования высокого уровня, или алгоритмическим языкам (т. е. языкам, специально разработанным для записи алгоритма). Подобно другим алгоритмическим языкам в Бейсике используются отдельные слова английского языка и простая математическая символика. Английское слово BASIC является аббревиатурой от Beginner's All—purpose Symbolic Instruction Code — универсальный код символических инструкций для начинающих. Первоначальная версия языка была создана в Дартмутском колледже (США) в 1965 г. специально для того, чтобы облегчить непрофессиональным программистам общение с ЭВМ.

Несмотря на весьма значительные последующие расширения языка, его отличительными чертами остались ориентация на диалоговый процесс подготовки и выполнения программ на ЭВМ, простота в изучении и применении. В настоящее время Бейсик — один из наиболее распространенных языков программирования общего назначения.

Прежде чем переходить к изучению Бейсика микроЭВМ «Искра 226», следует отметить, что дополнения и расширения Бейсика при реализации его на различных ЭВМ зачастую осуществлялись независимо друг от друга. Вследствие этого различные версии языка, сохраняя общее первоначальное «ядро», могут существенно отличаться друг от друга как синтаксисом, так и семантикой языковых средств. Бейсик 02 микроЭВМ «Искра 226» — одна из наиболее мощных версий языка. Она включает весьма разнообразные и гибкие средства программирования широкого круга прикладных задач, удобные как для начинающих, так и для квалифицированных программистов.

Символика Бейсика микроЭВМ «Искра 226» включает прописные и строчные буквы латинского (от A до Z и от a до z) и русского (от А до Я и от а до я) алфавитов, цифры (от 0 до 9) и специальные знаки (символы арифметических действий, синтаксические разделители и т. п.)[1].

Основным синтаксическим элементом языка является *оператор*— команда, предписывающая ЭВМ выполнение некоторой операции. Оператор является минимальной языковой единицей, которую «понимает» система, т. е. может выполнять соответствующие действия.

Каждый оператор содержит одно или несколько *ключевых слов*— строго определенную последовательность букв, по которой система распознает предписываемую операцию. В качестве ключевых слов, как правило, используются отдельные слова английского языка. Ключевое слово может дополняться строго определенными для каждой операции параметрами, конкретизирующими операцию. В качестве параметров выступают данные, с которыми должна быть осуществлена операция. Данные могут присутствовать в явном виде (числа, последовательность символов), в виде переменных, функций или их комбинаций — так называемых выражений. Рассмотрим следующие примеры операторов.

Оператор ввода значений переменных с клавиатуры состоит из ключевого слова INPUT, текста подсказывающего сообщения, которое может быть выдано на экран (необязательный параметр) и списка имен переменных, которым должны быть присвоены значения при выполнении оператора. Этот оператор предписывает машине выдать на экран подсказывающее сообщение и знак вопроса, затем ожидать, пока на клавиатуре не будут набраны значения переменных и нажата клавиша CR/LF. Например, в операторе

```
INPUT A
```

указано имя числовой переменной A, т. е. переменной, значением которой может быть число. Этот оператор предписывает набранное на клавиатуре число присвоить переменной A (иначе говоря, занести число в поле оперативной памяти, отведенное системой под значение переменной A).

Оператор вывода на печать начинается с ключевого слова PRINT, за которым следует список выражений, значения которых должны быть напечатаны. Например, оператор

```
PRINT A
```

печатает на экране число—значение переменной A, а оператор

```
PRINT  A*0.03
```

[1] Полный перечень символов приведен в приложении 1.

— результат умножения значения переменной А на 0.03,

Примечание. В Бейсике для записи десятичных дробей используется точка, а в качестве знака умножения — * (звездочка).

Оператор Бейсика или последовательность из нескольких операторов, разделенных двоеточием (так называемая строка непосредственного счета), могут выполняться непосредственно после ввода в машину. Это используется при простейших разовых вычислениях (см. режим непосредственного счета в гл. 2) или при управлении работой машины (очистка памяти, запуск программы, просмотр дискового каталога и т. п.).

Программа, написанная на Бейсике, представляет собой последовательность нумерованных строк, каждая из которых содержит один или несколько разделенных двоеточием операторов. Операторы программы выполняются в порядке возрастания нумерации строк.

Рассмотрим пример простейшей программы с использованием указанных операторов.

Пример 3.1

```
1 INPUT A
2 PRINT A*0.03
```

Эта программа состоит из двух программных строк с номерами 1 и 2. Строка 1 содержит оператор ввода, предписывающий принять с клавиатуры числовое значение переменной А, а строка 2 — оператор печати, предписывающий напечатать результат умножения значения переменной А на 0.03. Программа будет выполняться в порядке возрастания номеров строк, т. е. сначала будет принято значение переменной, а затем будет вычислено и напечатано произведение. Заметим, что эта программа эквивалентна следующей программе, состоящей из одной строки

```
1 INPUT A: PRINT A*0.03
```

Некоторые операторы Бейсика могут изменять нормальную последовательность выполнения операторов, вызывая переход к выполнению строки, указанной в операторе. Например, так называемый оператор условного перехода IF-THEN (ЕСЛИ-ТО) вызывает переход к выполнению строки, номер которой указан после слова THEN, но только в том случае, если выполняется условие, указанное после слова IF. Рассмотрим следующую программу[1].

Пример 3.2

```
1 INPUT "ЧИСЛО", A
2 PRINT "КУБ=;" A^3
3 INPUT "ПРОДОЛЖИТЬ? (Д/Н)", A⊙
4 IF A⊙="Д" THEN 1
```

Данная программа принимает вводимое с клавиатуры значение и присваивает его числовой переменной А (строка 1). При этом текст, заключенный в кавычки, при выполнении операции выдается на экран в качестве подсказки перед вопросительным знаком

ЧИСЛО?

Затем, при выполнении строки 2, печатается текст

КУБ=

и сразу после него (определяется символом-разделителем «;») — значение куба введенного числа (символ /\ обозначает операцию возведения в степень). После этого, при выполнении строки 3, запрашивается значение символьной переменной А⊙. Набранная последовательность символов присваивается переменной (иначе говоря, заносится в поле памяти, выделенное под значение этой переменной).

Если выполняется условие, оговоренное в операторе условного перехода строки 4, т. е. в качестве значения переменной А⊙ вводится символ Д, то осуществляется переход к строке 1, и вся программа выполнится сначала. В противном случае выполнение программы прекращается, поскольку за оператором условного перехода больше операторов нет.

В терминологии «Искры 226» от операторов Бейсика следует отличать команды, которые служат для непосредственного управления работой системы с клавиатуры. Каждой команде соответствует определенная управляющая клавиша. При их нажатии не высвечивается наименование команды (клавиши), а сразу выполняется соответствующее действие. Эти команды не могут быть включены в текст программы.

[1] По техническим причинам знак ¤ в тексте книги заменен знаком ⊙

В качестве примера можно привести команды CR/LF и RESET. Клавиша возврата каретки CR/LF нажимается по окончании набора на клавиатуре значений переменных, строк операторов или программных строк (названа так по аналогии с соответствующей клавишей пишущей машинки). По этой команде система принимает набранную информацию и осуществляет с нею необходимые действия. По команде RESET прекращается выполнение операторов и очищается экран.

3.2. Ввод текста программы в ЭВМ

Для того чтобы ЭВМ могла выполнить программу, последняя должна быть предварительно введена в ее оперативную память. Текст программы вводится в машину построчно в режиме диалога с использованием клавиатуры и дисплея. Как и ввод строк непосредственного счета, ввод программных строк в машину может быть осуществлен только после сообщения машины «:» в начале строки дисплея, означающего, что машина находится в ожидании приема управляющей информации.

Пользователь может осуществлять ввод программных строк сразу после появления на экране сообщения о готовности ЭВМ к работе

```
READY   BASIC   02  05.10.84

:_
```

в котором «05.10.84» — дата последней редакции системного программного обеспечения. Если на машине уже работали, то перед вводом программного текста желательно очистить память с помощью оператора очистки памяти CLEAR. Для этого следует набрать слово CLEAR (по буквам или же одним нажатием клавиши с этим словом)

```
: CLEAR_
```

и затем нажать клавишу возврата каретки CR/LF. Экран и память машины очистятся, и на экране снова появится сообщение о готовности.

Для очистки только экрана без изменения содержимого памяти ЭВМ может быть использована клавиша сброса RESET.

При вводе программной строки пользователь должен сначала набрать ее номер, а затем один или несколько операторов, разделяя их двоеточием. Операторы Бейсика могут набираться как по буквам, так и целым словом (если для набора данного оператора имеется клавиша на клавиатуре), например

```
:1   INPUT A
```

Для набора операторов можно использовать только прописные буквы английского алфавита. Пробелы внутри программной строки машиной игнорируются, обычно их набирают только для удобства последующего чтения программного текста. Если набираемая строка выходит за пределы строки экрана (80 символов), то курсор автоматически переходит в начало следующей строки. Однако общая длина вводимой строки не должна превышать 253 символа.

Набранная строка вводится в оперативную память нажатием клавиши возврата каретки CR/LF. Строка, начинающаяся числом, воспринимается системой как программная и не исполняется сразу, а заносится в оперативную память под номером, соответствующим этому числу. При этом проверяется соответствие набранной строки правилам синтаксиса языка Бейсик. В случае, если строка синтаксически правильна, на экран выводится сообщение «:», означающее, что можно продолжать вводить программные строки или осуществлять другие функции управления.

```
:1   INPUT  A
:_
```

Если в строке обнаружена синтаксическая ошибка, то на экран выдается сообщение об ошибке: текст введенной строки и ниже символ /\, указывающий на позицию строки, начиная с которой обнаружено отклонение от синтаксиса языка, а также цифровой код ошибки. Например, при попытке ввести программную строку

```
:2 PRINT A*
```

с незаконченным оператором пользователю будет выдано следующее сообщение об ошибке:

```
:2  PRINT A*
2  PRINT  A*
             /\ERR 06
:_
```

(код 06 — синтаксическая ошибка при вводе строки с клавиатуры).

Введенная строка, содержащая ошибку, может быть заменена путем повторного ввода правильной строки с тем же самым номером.

Номером программной строки может быть любое число от 0 до 9999 включительно. При этом строки программы могут вводиться в любом порядке: при выполнении учитываются только номера программных строк независимо от последовательности их ввода. При первоначальном вводе удобно присваивать строкам номера, начиная с 10 и далее с шагом 10. Это позволяет при последующих модификациях программы вводить дополнительные строки с промежуточными номерами. Для большего удобства набора номеров строк на клавиатуре микроЭВМ имеется клавиша STMT NUMBER, при каждом нажатии которой автоматически набирается номер строки на 10 больше максимального номера строки программы, содержащейся в памяти.

Рассмотрим пример. Сначала очистим оперативную память, набрав оператор CLEAR

```
: CLEAR_
```

и нажав клавишу CR/LF. Теперь в памяти нет программных строк и после нажатия клавиши STMT NUMBER появится число 10.

```
:10__
```

Наберем текст строки 10 и, нажав клавишу CR/LF, введем ее в память

```
:10 INPUT A
:__
```

Теперь при следующем нажатии клавиши STMT NUMBER будет введен номер на 10 больше предыдущего, т. е. 20

```
:10  INPUT A
:20__
```

и т. д.

Пока строка еще не введена в память, т. е. до нажатия клавиши CR/LF, она может быть скорректирована с помощью клавиши BACKSPACE или стерта нажатием клавиши LINE ERASE. При однократном нажатии клавиши BACKSPACE курсор передвигается на одну позицию влево, а находившийся в этой позиции символ стирается. Например, в строке

```
:20  PRINT  A*0.03__
```

после двух нажатий клавиши BACKSPACE будут стерты два последних символа

```
:20  PRINT  A*0.
```

При нажатии клавиши LINE ERASE стирается вся строка.

Все строки программы должны иметь различные номера. Если номер вводимой программной строки совпадает с номером какой-либо строки, уже имеющейся в памяти, эта строка замещает в памяти строку, введенную ранее. Пусть, например, сначала была набрана строка с номером 20

```
:20 PRINT A/\3
:__
```

Затем другая строка с тем же номером

```
:20  PRINT  A*0.03
:__
```

В результате строка с оператором PRINT A*0.03 заменит в памяти ранее введенную строку PRINT A/\3. Таким способом можно заменить строки, содержащие какие-либо ошибки. Если, например, ошибочно была введена строка, содержащая синтаксическую ошибку

```
:20  PRINT  A*
20  PRINT A*
                /\ERR 06
:__
```

Для ее исправления достаточно набрать верный вариант строки

```
20  PRINT  A*0.03
:__
```

В результате ошибочная строка 20 будет заменена в памяти на вновь введенную.

При вводе пустой программной строки, т. е. наборе номера строки и нажатии клавиши CR/LF, строка с таким номером удаляется из оперативной памяти. Например, для удаления строки

```
:30 PRINT A/\0.03
```

достаточно набрать 30 и нажать клавишу CR/LF.

```
:30
:_
```

3.3. Редактирование строк

В большинстве случаев гораздо удобнее не заново вводить строки вместо ошибочных, а лишь изменять введенные ранее. Для этого на клавиатуре существуют специальные клавиши редактирования.

Перед тем, как приступить к редактированию программной строки, уже занесенной в память, следует набрать номер строки, которая должна быть отредактирована, и нажать клавишу EDIT. В начале строки экрана вместо сообщения : появится сообщение * , означающее, что система находится в режиме редактирования строк. Теперь необходимо нажать клавишу RECALL для вызова на экран текста строки, предназначенного для редактирования. В режиме редактирования можно дополнительно пользоваться следующими командами, облегчающими внесение изменений в текст строки:

←	— переместить курсор влево на 1 позицию;
← — — — —	— переместить курсор влево на 5 позиций;
→	— переместить курсор вправо на 1 позицию;
— — — — →	— переместить курсор вправо на 5 позиций;
↑	— переместить курсор вверх на 1 строку экрана;
↓	— переместить курсор вниз на 1 строку;

INSERT —вставить пробел в текущую позицию строки, сдвинув вправо правую часть строки, начиная с текущего символа;

DELETE — удалить символ из текущей позиции строки, сдвинув влево правую часть строки;

ERASE — удалить правую часть строки, начиная с текущей позиции.

По завершении редактирования необходимо нажать клавишу CR/LF для занесения измененной строки в память ЭВМ. Рассмотрим пример, где требуется заменить набранную ранее строку

```
20 PRINT A*0.03
```

на строку

```
20 PRINT A/\3
```

путем редактирования исходной строки. Система находится в ожидании управляющих сообщений оператора

```
:_
```

Сначала наберем номер строки, подлежащей редактированию,

```
:10_
```

Для перехода к режиму редактирования нажмем клавишу EDIT

```
*10_
```

и нажатием клавиши RECALL вызовем на экран строку 10

```
*10 PRINT A*0.03_
```

При редактировании сначала заменим символ * на /\. Для этого подведем курсор под *, нажмем клавишу перемещения курсора на 5 позиций влево ←————

```
*10 PRINT A*0,03
```

и клавишу /\

```
*10 PRINT A/\0.03
```

Затем, нажав клавишу DELETE, сотрем символ над курсором, т. е. 0, и сдвинем оставшуюся часть строки на освободившееся место

```
* PRINT A/\ :03
```

Нажмем клавишу DELETE еще два раза

```
*10 PRINT A/\3
```

Редактирование закончено, для занесения отредактированного варианта строки в память следует нажать клавишу CR/LF

```
*10  PRINT  A/\3
:__
```

В режиме редактирования рассмотренная ранее клавиша BACKSPACE функционирует аналогично клавише ←, т. е. перемещает курсор на 1 позицию влево без стирания. Клавиша LINE ERASE, как и ранее, стирает всю редактируемую строку. При этом символ * в начале строки заменяется на :, т. е. система выходит из режима редактирования.

В режиме редактирования можно осуществлять объединение нескольких ранее набранных строк программы в одну. Для этого следует вызвать на экран для редактирования первую из объединяемых строк, в конце строки набрать : и номер следующей присоединяемой строки. После нажатия клавиши RECALL копия текста этой второй строки присоединяется к тексту первой строки (после двоеточия). При этом сама присоединенная строка останется в памяти без изменений. Для соединения большего числа строк этот процесс может быть продолжен. Нажатием клавиши CR/LF объединенная строка будет введена в оперативную память. Оригиналы присоединенных строк, если в них нет необходимости, могут быть стерты.

Рассмотрим пример объединения строк. Пусть необходимо объединить две набранные строки программы в одну (см. пример 3.1):

```
:1  INPUT  A
:2   PRINT   A*0.03
```

Наберем номер первой строки и нажмем клавиши EDIT и RECALL:

```
*1 PRINT  A__
```

Затем наберем : и номер присоединяемой строки

```
*1   PRINT   A:2__
```

После нажатия клавиши RECALL получим объединенную строку

```
*1   INPUT  A:   PRINT  A*0.03__
```

Для ее ввода необходимо нажать клавишу CR/LF

```
*1   INPUT  A:   PRINT  A*0.03
:__
```

В результате строка 1 будет заменена на объединенную, а строка 2 останется неизменной.

Клавишами режима редактирования можно пользоваться и при наборе строк (программных или непосредственного счета), т. е. еще до того, как нажата клавиша CR/LF. В этом случае для перехода к режиму редактирования нужно только нажать клавишу перехода к режиму редактирования EDIT.

Пусть, например, нужно набрать строку непосредственного счета

```
:PRINT  20.12+2.87__
```

Еще до завершения набора строки обнаружилась ошибка: вместо 20.12 должно быть набрано 2.12. После нажатия клавиши EDIT сообщение : в начале строки заменяется на * —индикатор режима редактирования

```
*PRINT   20.12 + 2.87__
```

Теперь с помощью клавиш ←———— и ← подведем курсор под цифру 0

```
*PRINT   20.12+2.87
```

Нажав клавишу DELETE, удалим символ под курсором, т. е. 0

```
*PRINT  2_.12 + 2.87
```

С помощью клавиш ————→ и → вернем курсор в конец строки

```
*PRINT   2.12+2.87_
```

Теперь можно продолжить набор строки

```
*PRINT  2.12+2.87+5.5__
```

и выполнить ее, нажав клавишу CR/LF,

```
*PRINT 2.12 + 2.87 + 5.5
```

```
10.49
:__
```

Как уже отмечалось, строки непосредственного счета не хранятся системой, однако последняя введенная строка (независимо от того, была ли она выполнена или было выдано сообщение об ошибке) может быть вновь вызвана на экран для редактирования и повторного ввода. Для этого необходимо нажать клавиши EDIT и RECALL.

Допустим, была набрана строка непосредственного счета

```
:PRINT 4.2/\__
```

и нажата клавиша CR/LF

```
PRINT   4.2/\
/\ERR 06
:__
```

После нажатия клавиш EDIT

```
*__
```

и RECALL

```
*PRINT 4.2/\__
```

ранее введенная строка может быть исправлена с использованием клавиш режима редактирования

```
*PRINT 4.2/\3__
```

и выполнена по команде CR/LF

```
74.088
:__
```

3.4. Вывод текста программы

Для вывода текста программы, находящейся в оперативной памяти, на экран используется оператор LIST. Чтобы осуществить выдачу текста программы, достаточно выполнить оператор LIST в режиме непосредственного счета, т. е. набрать после сообщения системы : ключевое слово LIST и нажать клавишу CR/LF. По оператору LIST текст программы всегда выводится в порядке возрастания номеров строк.

Допустим, была набрана программа (см. пример 3.2)

```
: 10    INPUT "ЧИСЛО", A
: 20    PRINT "КУБ ="; A^3
: 30    INPUT "ПРОДОЛЖИТЬ? (Д/Н)", A⊙
: 40    IF A⊙="Д" THEN 10

: LIST
10  INPUT "ЧИСЛО", A
20  PRINT "КУБ = "; A^3
30  INPUT "ПРОДОЛЖИТЬ? (Д/Н)", A⊙
40  IF A⊙=  "Д" THEN 10
```

Для выдачи на экран желаемой части программы используются различные модификации оператора LIST. Если после слова LIST указать номер строки и запятую, то на экран будет выдан программный текст, начиная с указанного номера строки. В данном случае

```
:LIST 20,
20  PRINT "КУБ ="; A^3
30 INPUT "ПРОДОЛЖИТЬ? (Д/Н)", A⊙
40 IF A⊙="Д" THEN 10
```

Примечание. Текст программы выдается без символа «:» в начале строки, поскольку, как уже отмечалось, сообщение «:» выдается системой только при пошаговом вводе строк.

Если указан номер строки и через запятую еще один номер строки, то на экран выдается программный текст между двумя номерами

```
:LIST ,30
10 INPUT. "ЧИСЛО", A
20 PRINT "КУБ ="; A^3
30 INPUT "ПРОДОЛЖИТЬ? (Д/Н)",A⊙
```

Если после слова LIST следует запятая и затем номер строки, то на экран выводится программный текст с начала программы и до строки с заданным номером

```
:LIST 20,30
20  PRINT  "КУБ   = ";   A/\3
30   INPUT  "ПРОДОЛЖИТЬ?   (Д/Н)",   A⊙
:_
```

Если программа или заданный ее фрагмент не помещаются на 24 строках экрана, то после выполнения оператора на экране остаются последние программные строки, умещающиеся на экране. Для покадрового просмотра на экране программного текста, не умещающегося на экране, используется оператор LIST с параметром S, который должен следовать непосредственно после слова LIST, например

```
LIST S
LIST S 10, 120
LIST S, 50
```

и т. п. При выполнении оператора LIST с параметром S на экран сначала выдаются первый кадр (в 23 строки экрана) специфицированного в операторе программного текста и сообщение : . После этого для просмотра каждого следующего кадра следует нажимать на клавишу CR/LF.

3.5. Исполнение программ

Запуск программы, находящейся в оперативной памяти ЭВМ, осуществляется с помощью оператора RUN. Оператор RUN обычно вводится в ЭВМ в режиме непосредственного счета, т. е. после сообщения : . После набора слова RUN и нажатия клавиши CR/LF начинается выполнение программы со строки с наименьшим номером.

Рассмотрим выполнение программы из примера 3.1.

```
:1  INPUT
:2 PRINT  A*0.03
:RUN
```

При выполнении оператора в строке 1 на экран выдается, с?» и система ожидает ввода числа с клавиатуры

```
?__
```

Набираемое число высвечивается на экране

```
?25__
```

Нажатием клавиши CR/LF набранное значение присваивается переменной A, а оператор в строке 2 печатает результат умножения этого значения на 0.03:

```
?25
.75
```

После запуска программы из примера 3.2 на экран будет выдано подсказывающее сообщение «ЧИСЛО» со знаком вопроса и система будет ожидать ввода числа пользователем.

```
:10 INPUT "ЧИСЛО", A
:20 PRINT "КУБ =";A/\3
:30 INPUT "ПРОДОЛЖИТЬ? (Д/Н)".A⊙
:40 IF A⊙="Д" THEN 10
:RUN
ЧИСЛО?_
```

Пользователь должен набрать на клавиатуре число

```
ЧИСЛО? 12__
```

и нажать клавишу CR / LF. Система выдаст слово «КУБ = », результат возведения введенного числа в куб, а затем подсказывающее сообщение «ПРОДОЛЖИТЬ? (Д/Н)» со знаком вопроса и будет ожидать

ввода ответа

```
КУБ= 144
ПРОДОЛЖИТЬ? (Д/Н)?__
```

Если пользователь хочет продолжить вычисления, он должен набрать «Д»

```
ПРОДОЛЖИТЬ? (Д/Н)?Д_
```

и нажать клавишу CR/LF. Выполнение программы продолжится с начала

```
ЧИСЛО? 13
КУБ= 169
ПРОДОЛЖИТЬ? (Д/Н)?__
```

Если пользователь ответит «Н», то выполнение программы закончится

```
ПРОДОЛЖИТЬ?  (Д/Н)?Н__
:__
```

После слова RUN допускается указание номера строки, с которой в этом случае начнется выполнение программы. Например, в случае рассмотренной программы

```
:RUN 30
ПРОДОЛЖИТЬ?  (Д/Н)?__
```

и т. д.

Заметим, что выполнение оператора запуска программы с номером строки приведет к ее исполнению, даже если в тексте программы есть неправильно набранные строки. Естественно, выполнение программы прекратится, как только встретится строка с неверным оператором,

Глава 4. ОСНОВНЫЕ ОПЕРАТОРЫ ЯЗЫКА БЕЙСИК

4.1. Арифметические выражения

С помощью операторов языка Бейсик можно описывать операции с данными. Числовые данные в Бейсике могут быть заданы постоянными значениями — константами, представляемыми в виде обычных десятичных чисел, например

```
25
2014.35
0.18
.0125
– 1.2
+.85
```

и т. д.

Примечание. Незначащие нули и знак + для положительных чисел могут быть опущены, а в десятичных дробных числах в качестве разделителя, как уже отмечалось, используется точка.

Диапазон десятичных чисел, которые могут обрабатываться в микроЭВМ «Искра 226», —от 10^{-99} до $9.99999999 \cdot 10^{99}$. Для записи числовых констант можно использовать не более 13 цифр, а также знак и десятичную точку.

При представлении очень больших или очень малых чисел вместо их естественной формы используется так называемая экспоненциальная форма представления чисел. Число представляется как числовая константа в естественной форме, умноженная на целую степень числа 10. При этом сначала записывается мантисса, затем служебный символ Е и собственно показатель степени числа 10. Например,

```
12E12
–.1E – 80
4.5E+9
```

и т. п. Экспоненциальная форма 12E12 представляет число $12 \cdot 10^{12}$, т. е. 12 000 000 000 000. Порядок чисел, представленных в экспоненциальной форме, должен быть целым числом от —99 до 99 (знак + не обязателен).

Если в естественной форме числовая константа содержит не более 13 десятичных разрядов, то она

может быть представлена и в естественной и в десятичной форме. Например,

```
:PR1NT  0.00123*252
.30996
: PRINT   1.23E—3*252
.30996
:—
```

Для представления числовых данных наряду с числовыми константами в операторах Бейсика используются так называемые числовые переменные, которые могут принимать различные числовые значения. В качестве имени числовой переменной можно использовать любую букву латинского алфавита от A до Z или букву с цифрой от 0 до 9. Таким образом, в одной программе возможно до 286 различных переменных

```
A, A0, A1, A2,   .., A9, W, W0, W1, ..., Z9
```

Следовательно, в операторах Бейсика в качестве числовых данных наряду с числовыми константами могут быть указаны имена числовых переменных, например

```
PRINT A/2
PRINT  W0+S/D9—2
```

При этом в операторах вместо конкретных чисел — констант— указываются ссылки на поля памяти, с содержимым которых должна быть выполнена соответствующая операция.

Поскольку в микроЭВМ «Искра 226» действия с переменными, которым приписываются только целые значения, могут быть осуществлены быстрее, чем с действительными числами, а для хранения значений целых переменных требуется меньше памяти, в язык введены целочисленные переменные. В качестве имени целочисленных переменных может использоваться буква латинского алфавита или буква с цифрой, к которым добавляется символ %

```
A%, A0%, A1%, A2%, .... A9%,  W%,    W0%,   W1%, .... Z9%
```

Допустимый диапазон значений целочисленных переменных — от —7999 до 7999.

В Бейсике, кроме того, определен ряд функций (числовых и не числовых аргументов), значением которых являются числа. Таким образом, в качестве числовых данных в операторах Бейсика могут выступать числовые константы, переменные, функции, а также арифметические выражения — комбинации числовых констант, переменных и функций с использованием следующих служебных символов арифметических действий: + (сложить), — (вычесть), * (умножить), / (разделить), /\ (возвести в степень), ((открывающая скобка),) (закрывающая скобка).

Примеры арифметических выражений:

```
2+3/A
13–A/\ (5–1.2)
SIN(30)/\2
2E10*152/14.36
A+B0/\(18/\2)
16/X+13
```

Значением арифметического выражения является число. При его вычислении соблюдаются следующие приоритеты в очередности выполнения арифметических действий:

вычисление значений выражений в скобках;

возведение в степень;

умножение и деление;

сложение и вычитание,

Например,

```
:PRINT  5+3*2/\3
29
:PRINT  5+(3*2)/\3
221
:—
```

4.2. Строки алфавитно-цифровых символов

Операторы языка Бейсик позволяют предписывать машине действия не только с числами, но и с

текстами, точнее говоря, с последовательностями (строками) алфавитно-цифровых символов (букв, цифр и специальных знаков) Бейсика, включая все прописные и строчные буквы латинского и русского алфавитов.

В операторах Бейсика символьные данные, как и числовые, могут задаваться в виде констант, т. е. непосредственно в виде строк алфавитно-цифровых символов, которые при этом должны быть обязательно заключены в кавычки или апострофы. Например,

```
"ИТОГОВАЯ СУММА"
'РАСЧЕТ'
"BASIC  02"
"15.03.85"
```

Следует помнить, что если строка заключена в кавычки или в апострофы, то внутри нее не должно быть кавычек или апострофов. Например,

Неправильно	*Правильно*
"МИКРОЭВМ "ИСКРА 226" "	"МИКРОЭВМ 'ИСКРА 226' "
или	или
'МИКРОЭВМ 'ИСКРА 226' '	'МИКРОЭВМ "ИСКРА 226" '

Строки символов, заключенные в кавычки, распознаются (и в последующем выводятся) системой как строки, состоящие из прописных букв соответствующего алфавита, а строки, заключенные в апострофы,— как строки, состоящие из строчных букв. Например,

```
:  PRINT  "ДОКУМЕНТ"
ДОКУМЕНТ
:  PRINT  'ДОКУМЕНТ'
документ
:
```

Таким способом можно задавать символьные константы из строчных букв, хотя на клавиатуре имеются только клавиши для набора прописных букв.

В качестве примера использования символьных констант в программе можно привести программу из примера 3,2:

```
:10    INPUT "ЧИСЛО", A
:20    PRINT "КУБ *";A^3
:30    INPUT "ПРОДОЛЖИТЬ?(Д/Н)",A⊙
:40    IF A⊙="Д" THEN 10
```

Символьные данные могут быть также представлены в виде символьных переменных, значениями которых являются символьные строки. Именем символьной переменной может быть буква латинского алфавита или буква с цифрой от 0 до 9 с символом ⊙. Таким образом, допускаются следующие имена символьных переменных:

A⊙, A0⊙, A1⊙, A2⊙, ... , A9⊙, B⊙, B0⊙, B1⊙, ... ,Z9⊙

Кроме того, в качестве символьных данных может выступать функция STR, значением которой является символьная строка (гл. 14).

Для хранения значения символьной переменной в памяти автоматически выделяется поле памяти размером 16 символов. Это означает, что, если в программе нет дополнительных указаний, максимальная длина строки, которая может являться значением символьной переменной, составляет 16 символов, включая внутренние пробелы.

При необходимости с помощью оператора DIM можно задавать произвольную максимальную длину значения символьной переменной в пределах от 1 до 253 символов. В операторе после слова DIM должно следовать имя символьной переменной и за ним — число, означающее максимальную длину значения данной переменной. Например, оператор

```
DIM  A⊙20
```

обеспечивает резервирование поля памяти размером 20 символов под значение символьной переменной A⊙.

Во избежание недоразумений оператор DIM с заданием нестандартной длины переменной в программе должен обязательно стоять перед другими операторами, содержащими имя этой переменной.

В одном операторе DIM могут быть заданы длины сразу для нескольких переменных. Имена

переменных с указанием их длин в этом случае перечисляются через запятую

```
DIM  A⊙ 20, D⊙ 250, U⊙1
```

4.3. Присваивание значений переменным

Начальное значение числовых переменных — 0, символьных — пробел. Для того чтобы приписывать переменным какие-либо значения, можно использовать оператор присваивания значений LET. После слова LET указываются имя переменной, которой должно быть присвоено новое значение, затем символ присваивания = и собственно присваиваемое значение. Например, оператор

```
LET  A=3
```

предписывает присвоить значение 3 числовой переменной A, а оператор

```
LET A⊙="БЕЙСИК"
```

предписывает присвоить значение «Бейсик» символьной переменной A⊙.

Слово LET указывать не обязательно, т. е. приведенные выше операторы эквивалентны следующим:

```
A=3
A⊙,  "БЕЙСИК"
```

Если оператор LET используется для присваивания значений числовой переменной, то справа от знака = в общем случае может быть указано арифметическое выражение, например

```
X%=20+15*A+8*A/\2
P=(E+SIN(E))/\3
```

При выполнении оператора сначала осуществляется вычисление значения арифметического выражения справа от знака = и только затем вычисленное значение присваивается переменной слева от =. Поэтому в арифметическом выражении может использоваться та же переменная, что и слева от =, например

```
A=A/\2
```

Пример 4.1

```
:10 A=15
:20 A=A/\2
:30 PRINT A
:RUN
225
:__
```

Сначала переменной A присваивается значение 15, затем это значение возводится в квадрат и принимается за новое значение переменной.

Если с помощью оператора LET присваивается значение символьной переменной, то справа от знака = должна быть указана символьная строка, например

```
A⊙ = "РАСЧЕТ  ЦЕНЫ"
C⊙ = 'BASIC'
K⊙ = E⊙
```

С помощью одного оператора присваивания можно присвоить значение сразу нескольким переменным одного типа (либо символьным, либо числовым). Для этого имена переменных перечисляются через запятую слева от знака —. Например, последовательности операторов

```
X=1                    A⊙ = "СПРАВКА"
E=1                    B⊙ = "СПРАВКА"
K% = 1
```

равносильны соответственно операторам

```
X, E, K% = 1          A⊙,  B⊙ = "СПРАВКА"
```

При запуске программ с помощью оператора RUN без указания номера строки числовым переменным автоматически присваивается значение 0, а символьным — пробел. Если же в операторе RUN указан номер строки, с которой должно начаться выполнение программы, то переменные сохраняют значения, присвоенные им ранее.

Пример 4.2

Наберем программу печати квадрата значения переменной А

```
:10 PRINT A/\2
:__
```

В режиме непосредственного счета присвоим этой переменной значение 15

```
:A = 15
:__
```

и запустим программу

```
:RUN
0
:__
```

По оператору RUN переменной А присваивается нулевое значение. Еще раз присвоим переменной А значение 15 и запустим программу по оператору RUN, задав на этот раз номер строки

```
:A=15
:RUN 10
225
:__
```

Обнуления не произошло, переменная А сохранила значение, которое было присвоено ей до выполнения программы.

При использовании оператора присваивания значений символьным переменным следует помнить об ограничении размера символьных переменных (как уже отмечалось в 4.2, она может устанавливаться в пределах от 1 до 253 символов, а по умолчанию—равной 16). Если длина присваиваемого значения больше размерности символьной переменной, то крайние справа символы игнорируются.

Пример 4.3

После выполнения оператора

```
A⊙ = "РАСЧЕТ СЕБЕСТОИМОСТИ"
```

значением символьной переменной А⊙ станет строка из первых 16 символов строки «РАСЧЕТ СЕБЕСТОИМОСТИ», т. е. «РАСЧЕТ СЕБЕСТОИМ»

```
:10 A⊙ ="РАСЧЕТ СЕБЕСТОИМОСТИ"
:20 PRINT A⊙
:RUN
РАСЧЕТ СЕБЕСТОИМ
:__
```

Для нормальной работы программы в ней необходимо задать длину символьной переменной не менее 20 символов. Дополним программу оператором DIM и снова запустим ее

```
:5 DIM A⊙ 20
:RUN
РАСЧЕТ СЕБЕСТОИМОСТИ
:__
```

При использовании оператора LET присваиваемое значение включается непосредственно в текст программы. Для присваивания переменным значений, вводимых пользователем непосредственно в ходе выполнения программы, используется оператор INPUT с указанием имени числовой или символьной переменной, значение которой должно быть введено с клавиатуры при выполнении программы, например

```
INPUT A
INPUT L⊙
```

При выполнении оператора INPUT на экран выводится символ «?» и машина ожидает, когда на клавиатуре будет набрано соответствующее значение, которое высвечивается на экране после вопросительного знака.

Набираемое значение должно соответствовать типу переменной в операторе INPUT. В случае ввода числовой переменной набирается допустимая числовая константа, а если вводится символьная переменная, пользователь может набирать любую последовательность символов (с учетом максимальной длины значения соответствующей переменной).

По команде CR/LF набранное значение присваивается переменной и выполнение программы продолжается. Если никакого значения набрано не было, после нажатия клавиши CR/LF переменная сохраняет значение, присвоенное ей ранее.

Вводимая символьная строка не обязательно должна заключаться в кавычки или апострофы.

Пример 4.4

```
:10  INPUT  A⊙
:20  PRINT  A⊙
:RUN
?ОПИСЬ ДЕЛ
ОПИСЬ  ДЕЛ
:__
```

Следует, однако, иметь в виду, что при наборе строки без кавычек или апострофов все крайние пробелы слева игнорируются. Выполним программу еще раз

```
:RUN
?              ОПИСЬ  ДЕЛ
ОПИСЬ  ДЕЛ
:__
```

В случае же использования кавычек или апострофов в значение переменной можно вставлять крайние слева пробелы, например

```
:RUN
?"              ОПИСЬ ДЕЛ"
            ОПИСЬ ДЕЛ
:__
```

При наборе значения для стирания последнего набранного символа и всей строки можно пользоваться соответственно клавишами BACKSPACE и LINE ERASE. Нажатием клавиши EDIT машина переводится в режим редактирования: в начале набираемой строки появляется символ * и для редактирования набираемого значения можно пользоваться всеми клавишами режима редактирования (см. 3.3).

С помощью одного оператора INPUT можно программировать ввод значений сразу нескольких числовых и символьных переменных в любом сочетании, имена которых перечисляются через запятую, например

```
INPUT  A,K0,F9⊙,E
```

При выполнении оператора пользователь должен набирать все требуемые значения в том же порядке, разделяя их запятой

```
?1987,12,ЦЕНА,1
```

Если значение символьной переменной включает запятую, то для того, чтобы машина при вводе восприняла ее не как разделитель значений, а как элемент символьной строки, это значение следует набирать в кавычках или апострофах, например

```
?1987,12,"ЦЕНА,ТЫС.РУБ.",1
```

При выполнении оператора INPUT с несколькими переменными допускается вводить требуемые в порядке очередности. Иными словами, если пользователь набрал не все значения переменных, перечисленных в операторе, и нажал клавишу CR/LF, то машина, присвоив введенные значения, снова выводит знак вопроса и ожидает приема значений остальных переменных. Пользователь может продолжить ввод значений или отказаться от набора, нажав клавишу CR/LF. В последнем случае все остальные переменные сохранят старые значения и система перейдет к выполнению следующего оператора.

Пример 4.5

Введем следующую программу:

```
:10      INPUT      A⊙,B,C⊙,D,E⊙
:20      PRINT      A⊙;B
:30      PRINT      C⊙;D
:40      PRINT      E⊙;C*D
```

В строке 10 запрограммирован ввод значений пяти переменных— символьной, числовой,

символьной, числовой и символьной. В строке 20 — печать значений первой пары этих переменных в том же порядке, в строке — 30 — второй пары, в строке 40 печать значения последней символьной переменной и произведения числовых переменных. Выполним эту программу

```
:RUN
?ДЛИНА,147,ШИРИНА,89,ПЛОЩАДЬ
ДЛИНА 147 ШИРИНА 89
ПЛОЩАДЬ 13083
:__
```

Выполним программу повторно

```
:RUN
? __
```

На этот раз наберем только два первых значения

```
?"ЦЕНА,РУБ.",5.1 __
```

Число 5.1 набрано ошибочно вместо 25.1. Для исправления ошибки перейдем в режим редактирования, нажав клавишу EDIT

```
?"ЦЕНА,РУБ.",5.1__
```

Подведем курсор под 5, нажмем клавишу вставки пробела INSERT

```
?*"ЦЕНА,РУБ.",__5.1
```

и наберем 2

```
?*"ЦЕНА,РУБ.",25.1
```

Теперь можно вернуть курсор в правый край строки и продолжить ввод. Вместо этого нажмем клавишу CR/LF

```
?*"ЦЕНА,РУБ",25.1
?__
```

Машина ожидает ввод значений остальных трех переменных. Наберем, например, еще два значения и нажмем клавишу CR/LF

```
? "КОЛИЧЕСТВО,ШТ.",875
?__
```

Машина ожидает значение единственной оставшейся переменной, после ввода которого выполнение программы продолжится

```
?"СТОИМОСТЬ,РУБ."
ЦЕНА,РУБ. 25.1
КОЛИЧЕСТВО,ШТ.  875
СТОИМОСТЬ,РУБ.  21962.5
:__
```

Выполним эту программу еще раз

```
:RUN
?ДЛИНА,147,ШИРИНА__
```

Теперь нажмем клавишу CR/LF

```
?__
```

Не набирая остальные значения, повторно нажмем клавишу CR/LF

```
?
ДЛИНА 147
ШИРИНА 0
0
:__
```

Машина сохранила начальные значения остальных переменных (0 для числовых и пробел для символьных переменных).

В операторе INPUT можно также задать текст сообщения, которое в качестве подсказки будет выдаваться пользователю при выполнении оператора (перед знаком вопроса). Текст в виде символьной

константы, т. е. строки символов в кавычках, помещается между словом INPUT и списком переменных. После символьной константы должна следовать запятая. Например,

```
INPUT "ЧИСЛО" K
```

При выполнении этого оператора будет выдана подсказка

```
ЧИСЛО ?
```

Использование подсказок значительно облегчает работу с программой.

Пример 4.6

Приведенная ниже программа позволяет возвести заданное число в заданную степень

```
:10 INPUT "ЧИСЛО", A
:20 INPUT "ПОКАЗАТЕЛЬ СТЕПЕНИ", E
:30 PRINT A^E
```

Выполним ее

```
:RUN
ЧИСЛО? 8
ПОКАЗАТЕЛЬ СТЕПЕНИ? 3
512
:_
```

Для ввода строк символов удобно пользоваться оператором LINPUT. В отличие от оператора INPUT в операторе LINPUT задается только одна символьная переменная, например

```
LINPUT A⊙
LINPUT "ВВЕДИТЕ ТЕКСТ", B⊙
```

При выполнении этого оператора машина автоматически переходит в режим редактирования, причем на экране появляется признак режима редактирования — символ * и высвечивается текущее значение специфицированной переменной. Таким образом, пользователь может не вводить значение переменной полностью заново, а отредактировать присвоенное ранее значение. При редактировании перемещение курсора за пределы максимальной длины символьной переменной автоматически блокируется.

Набранная или отредактированная строка вводится в память нажатием клавиши CR/LF. До нажатия этой клавиши на любом этапе редактирования можно вернуть исходное значение переменной с помощью клавиши RECALL.

Если перед именем символьной переменной в операторе указан символ —, то при выполнении оператора символ * перед значением символьной переменной не высвечивается.

4.4. Вывод данных

Для вывода данных на экран используется оператор печати PRINT. За словом PRINT должны следовать один или несколько элементов оператора PRINT. В качестве элемента может выступать арифметическое выражение или строка символов, например

```
PRINT A
PRINT A/\2–3+C2
PRINT "ЗАПИСКА"
PRINT W⊙
```

Если элементом PRINT является арифметическое выражение, то при выполнении оператора сначала вычисляется значение арифметического выражения и затем печатается полученное в результате вычислений число. При этом знак + перед положительными числами не печатается, вместо него выводится пробел, после числа также выводится пробел. Незначащие нули в целой части числа не выводятся. Числовые значения, находящиеся в пределах от 1E—12 до 1E13, выводятся в естественной форме, остальные — в экспоненциальной (см. 4.1), например

```
: PRINT 1.1999*10/\ —12
 .0000000000011
:_
```

При выводе символьных констант внешние кавычки или апострофы, в которые они заключены, не печатаются, например

```
: PRINT "ИСКРА 226"
```

```
ИСКРА 226
:__
```

Если элементов PRINT несколько, они разделяются запятой или точкой с запятой. Первый элемент оператора PRINT печатается, начиная с левого края строки экрана. Если в качестве разделителя используется точка с запятой, следующее значение печатается без перемещения курсора после печати предыдущего элемента (так называемый плотный формат печати).

Пример 4.7

Наберем и выполним следующую программу выдачи степеней введенного числа

```
:10  INPUT A
:20  PRINT A; A/\2; A/\3
:RUN
?—13
— 13  169  —2197
:__
```

Числа выведены вплотную друг за другом с учетом знака перед числом и пробела после числа.

Пример 4.8

Рассмотрим пример плотной печати символьных данных. Следующая программа поочередно принимает два символа, обозначающие число, два символа — месяц, два символа — последние две цифры года, а затем выводит их через точку, причем перед годом добавляется 19,

```
:10 DIM A⊙2.B⊙2.C⊙2
:20 INPUT "ЧИСЛО",A⊙
:30 INPUT "МЕСЯЦ",B⊙
:40 INPUT "ДВЕ ЦИФРЫ ГОДА",C⊙
:50 PRINT A⊙;".";B⊙;".";"19";C⊙
:RUN
ЧИСЛО?05
МЕСЯЦ?03
ДВЕ ЦИФРЫ ГОДА?77
05.03.1977
:__
```

Значения символьных переменных выводятся полностью с учетом крайних справа пробелов. Если бы в программе размерность переменных A⊙ и B⊙ не была объявлена равной двум (строка 10), то было бы напечатано

```
05                .03               .1977
```

поскольку по умолчанию их длины были бы приняты равными 16.

Если в качестве разделителя используется запятая (зонный формат), то следующее за ней значение печатается в начале новой 16-символьной зоны строки (каждая строка дисплея условно делится на 5 зон по 16 символов в каждой).

Пример 4.9

Заменим «;» в операторе PRINT программы примера 4.7 на «,»

```
:10  INPUT A
:20  PRINT  A,A/\2,A/\3
:RUN
?—13
—13
:___              169               —2197
```

Число —13 выведено в первой зоне, т. е. начиная с 1-й позиции строки экрана, 169 — в следующей, второй зоне, т. е. начиная с 17-й позиции (при этом перед положительным числом пропущена одна позиция вместо знака +), —2197 — в следующей, третьей зоне, т. е. начиная с 33 позиции.

При зонной печати в случае, если предыдущее значение печаталось в последней, пятой, зоне, следующее значение будет печататься в первой зоне следующей строки экрана.

При выводе в зонном формате можно пропускать зоны, ставя соответствующее число запятых подряд.

Пример 4.10

```
:10  INPUT  A
```

```
:20  PRINT  A,,A/\2,A/\3
:RUN
?—13
—13                    169        —2197
:
```

Значение переменной А печатается, как и ранее, в первой зоне. За переменной А в операторе PRINT стоят две запятые подряд, которые задают перемещение курсора в начало третьей зоны. В ней печатается квадрат значения переменной А, а в следующей четвертой — куб этого значения.

В одном операторе можно использовать оба разделителя в любом сочетании, а также комбинировать числовые и символьные данные.

Пример 4.11

```
:10  INPUT  A
:20  PRINT  A,  "КВАДРАТ = ";A/\2,"КУБ = ";A/\3
:RUN
?—13
—13                    КВАДРАТ =169            КУБ = —2197
:__
```

Значение А печатается с начала строки; в начале второй зоны печатается — строка символов «КВАДРАТ = » и вплотную к ней значение квадрата, в начале третьей зоны — строка символов «КУБ = » и вплотную к ней значение куба.

Плотный формат удобен для компактной выдачи числовой информации и слитных текстов. Зонный формат ориентирован на выдачу информации в табличной форме, если, конечно, в таблицах не более пяти колонок. При этом размер зоны таков, что в него вписываются число максимального формата и все значение символьной переменной стандартной длины.

Использование функции TAB в качестве элемента оператора PRINT позволяет перемещать курсор в любую позицию строки, находящуюся правее текущей позиции курсора. Функция TAB имеет следующий формат.

Примечание. Здесь и далее текст, заключенный в угловые скобки, обозначает наименование соответствующего синтаксического элемента. TAB (< аргумент >)

Аргументом функции TAB должно быть арифметическое выражение, например,

```
TAB  (38)
TAB  (A*2+3)
```

По функции TAB курсор перемещается из текущей позиции строки в позицию, номер которой равен целой части значения аргумента. При этом позиции строки нумеруются с нуля, т. е. крайняя слева позиция строки — 0-я, следующая—1-я и т. д. до последней 79-й. Например, оператор

```
PRINT  TAB(20);A
```

задает перемещение курсора из текущей позиции в 20-ю и печать в этой позиции значения переменной А.

Пример 4.12

```
:10  INPUT  A
:20  PRINT  A;TAB(9);"КВАДРАТ = ";A/\2;TAB(27);"КУБ = ";A/\3
:RUN
?—13
—13              КВАДРАТ=169                    КУБ= — 2197
:__
```

По оператору PRINT в строке 20 с позиции 0 выводится значение числа. Затем по функции TAB (9) курсор перемещается в позицию 9 и выводятся текст и значение квадрата А. После этого курсор перемещается в позицию 27 и выводятся текст и значение куба А.

Если позиция, задаваемая функцией TAB, находится левее позиции курсора в данный момент, то перемещения не произойдет. Если значение аргумента больше длины строки, то курсор установится на начало следующей строки.

Если в программе используются несколько операторов, то элементы первого оператора, как уже отмечалось, печатаются, начиная с левого края экрана. Начальная позиция печати каждого следующего оператора PRINT будет зависеть от того, чем закончилась запись предыдущего оператора PRINT:

если предыдущий оператор не закончился разделителем «,» или «;», то курсор переводится в начало

следующей строки экрана, т. е. вывод по новому оператору начнется с начала этой строки;

если предыдущий оператор закончился разделителем «;», то курсор остается на месте и вывод по следующему оператору начинается вплотную к предыдущему;

если предыдущий оператор закончился разделителем «,», курсор сдвигается в следующую зону и вывод по очередному оператору начинается со следующей зоны.

Оператор PRINT без параметров переводит курсор из текущего положения в начало следующей строки. Для стирания экрана и перевода курсора в левый верхний угол используется оператор PRINT с элементом HEX(03) (шестнадцатеричный код 03)

```
PRINT HEX(03)
```

Пример 4.13

```
:10  INPUT  A
:20  PRINT   HEX(03);  A
:30  PRINT
:40  PRINT  TAB(9);"КВАДРАТ,";A Л2,
:50  PRINT TA8(27);"КУБ=";A/\3
:RUN
?-13_
```

После нажатия клавиши CR/LF осуществится ввод набранного числа и начнет выполняться оператор PRINT в строке 20. Экран очистится, курсор переместится в левый верхний угол и напечатается введенное значение переменной А. По окончании выполнения . этого оператора курсор переместится в начало следующей строки. По оператору PRINT строки 30 осуществится перемещение курсора в следующую строку, т. е. пропуск строки. По оператору PRINT строки 40 напечатается квадрат числа и курсор переместится в начало следующей зоны текущей строки. По оператору PRINT строки 50 в этой зоне напечатается куб числа. В результате выполнения программы на экран будет выведено (начиная с левого верхнего угла)

```
– 13
КВАДРАТ =169                КУБ = —2197
:__
```

4.5. Операторы безусловного и условного переходов

Как отмечалось, программа на языке Бейсик выполняется в порядке возрастания номеров строк. Однако при программировании часто возникает необходимость изменить обычную последовательность выполнения операторов. Для этого служит ряд операторов, в том числе оператор безусловного перехода. Оператор безусловного перехода позволяет осуществить переход к выполнению строки с заданным номером из любого места программы.

Оператор состоит из слова GOTO и следующего после него номера строки, к которой должен быть осуществлен переход. Результатом выполнения оператора является переход к выполнению операторов строки с указанным номером. Например, при выполнении оператора

```
GOTO 20
```

будет осуществлен переход к выполнению строки 20 (если, конечно, строка с таким номером имеется в программе), независимо от того, где в программе находится этот оператор.

Оператор условного перехода также позволяет осуществить переход к выполнению строки с заданным номером, однако переход выполняется лишь в случае, если удовлетворяется некоторое условие, заданное в операторе. Иначе переход не осуществляется, и выполнение программы продолжается в обычном порядке, т.е. выполняется оператор, следующий за оператором условного перехода. Оператор условного перехода имеет следующий вид:

```
IF  <условие>  THEN  <номер строки>
```

где под условием подразумевается

$$< \text{а.в} > \left\{ \begin{array}{l} = \\ < \\ > \\ <> \\ <= \\ >= \end{array} \right\} < \text{а.в} >$$

(а.в. — арифметическое выражение)

или

< символьная строка > $\left\{ \begin{array}{l} = \\ < \\ > \\ <> \\ <= \\ >= \end{array} \right\}$ < символьная строка >

Примечание. Здесь и далее фигурные скобки означают, что должна быть выбрана одна из перечисленных в них синтаксических конструкций.

Использованные в этом определении символы отношения означают: = (равно), < (меньше), > (больше), О (не равно), <= (меньше или равно), >= (больше или равно).

Примеры:

A = 5 *

A>B + 4*2

X0<=Y

K< >5

A⊙ = "ДА"

M⊙< >E⊙

C⊙<"999"

При выполнении, допустим, оператора

```
IF  A = 5  THEN  20
```

если значение переменной А равно 5, будет осуществлен переход к выполнению строки с номером 20, а в противном случае будет выполнен следующий за оператором безусловного перехода оператор.

Заметим, что в некоторых случаях с помощью одного условия можно проверить сразу несколько условий. Например, последовательность операторов условного перехода, задающая переход к строке 1000, при выполнении хотя бы одного из условий

```
200  IF  A+B=0  THEN  1000
210  IF  Z=14*K  THEN  1000
220  IF  C=0  THEN  1000
230  IF  S=2  THEN  1000
```

равносильна следующему оператору:

```
200  IF  (A+B)*(Z—14*K)*C*(S—2)=0  THEN  1000
```

В случае символьных строк требуются некоторые разъяснения условий, включающих отношения «больше» или «меньше». Каждый символ символьной строки в ЭВМ представляется специальным числовым кодом. Все символы ранжируются в соответствии с этим кодом. При проверке условия в операторе условного перехода сравниваются сначала крайние слева символы сравниваемых строк. Если один из этих символов имеет больший ранг, то и соответствующая строка считается большей. Если же эти символы, имеют одинаковый ранг, то сравниваются между собой вторые слева символы и т. д. Например, строка

```
ИВАНОВ
```

«меньше», чем строка

```
ПЕТРОВ
```

поскольку код символа И меньше кода символа П (таблица кодировки символов приведена в приложении 1).

Рассмотрим пример программы вычисления доли в процентах с использованием операторов условного и безусловного переходов для предотвращения недопустимой операции деления на 0.

Пример 4.14

```
:10  INPUT  "ЧИСЛО  1",A
:20  INPUT  "ЧИСЛО  2",B
:30  IF B< > 0  THEN  60
:40  PRINT  "ЧИСЛО  2  НЕ ДОЛЖНО БЫТЬ РАВНЫМ 0"
:50  GOTO  20
:60  PRINT  A ;"СОСТАВЛЯЕТ";A/B* 100;"ПРОЦЕНТОВ ОТ" ; B
```

В строках 10 и 20 использован оператор ввода с клавиатуры значений числовых переменных A и B соответственно, в строке 30— оператор перехода по условию. Если значение переменной B не равно 0, будет осуществлен переход к выполнению строки 60, в которой вычисляется и печатается значение арифметического выражения A/B* 100, а также печатаются значения переменных A и B и пояснительный текст. Если значение переменной B равно 0, то условие в операторе строки 30 не будет удовлетворено и выполнится следующий оператор (в строке 40)—напечатается сообщение о недопустимости нулевого значения второго числа. После этого по оператору безусловного перехода в строке 50 осуществляется переход к строке 20, т. е. к повторному вводу с клавиатуры значения переменной B. Ниже приведены примеры диалога с программой.

```
ЧИСЛО 1?20
ЧИСЛО 2?200
  20 СОСТАВЛЯЕТ 40 ПРОЦЕНТОВ ОТ 200
:
ЧИСЛО 1?15
ЧИСЛО 2?
ЧИСЛО 2 НЕ ДОЛЖНО БЫТЬ РАВНО 0
ЧИСЛО 2?120
  15 СОСТАВЛЯЕТ 12.5 ПРОЦЕНТОВ ОТ 120
:_
```

При запуске программы с помощью оператора RUN без указания номера строки переменным присваиваются нулевые значения. Поэтому при повторном исполнении программы, когда пользователь не ввел второе число, было выдано сообщение о недопустимости нулевого значения и еще раз предложено ввести значение этого числа.

Теперь рассмотрим пример использования оператора условного перехода в случае символьных строк.

Пример 4.15

```
:10  INPUT  A⊙
:20  INPUT  B⊙
:30  PRINT
:40  IF A⊙>B⊙ THEN  80
:50  PRINT  A⊙
:60  PRINT  B⊙
:70  GOTO  100
:80  PRINT  B⊙
:90  PRINT  A⊙
:100  PRINT "СОРТИРОВКА  ЗАВЕРШЕНА"
```

В строках 10 и 20 — оператор ввода значений символьных переменных A⊙ и B⊙. После этого осуществляется пропуск строки экрана (оператор PRINT без параметров в строке 30). В строке 40 оператор условного перехода осуществляет переход к строке 80 в случае, если введенное значение A⊙ больше значения B⊙. При этом в строках 80 и 90 задана печать сначала значения B⊙ и в следующей строке экрана — значения A⊙. В строке 100 — оператор печати сообщения о завершении работы программы.

Если же условие оператора условного перехода в строке 40 не выполнено, т. е. значение A⊙ меньше значения B⊙ или равно ему, выполняется следующий по порядку оператор. Печатаются сначала значение A⊙ (строка 50) и затем значение B⊙ (строка 60), после чего осуществляется переход (строка 70) к печати сообщения о завершении работы программы в строке 100. Выполним эту программу, введя в качестве значений переменных A⊙ и B⊙ строки символов ИВАНОВ и ПЕТРОВ.

```
:RUN
?ИВАНОВ
?ПЕТРОВ
ИВАНОВ
ПЕТРОВ
СОРТИРОВКА ЗАВЕРШЕНА
:_
```

Теперь выполним эту программу еще раз, введя те же фамилии в обратном порядке

```
:RUN
? ПЕТРОВ
? ИВАНОВ
ИВАНОВ
ПЕТРОВ
СОРТИРОВКА ЗАВЕРШЕНА
:_
```

Как видим, программа выдает введенные значения в соответствии с рангом символов, т. е. осуществляет так называемую сортировку (упорядочение) при выдаче.

Примечание. Следует заметить, что в общем случае ранжировка символов русского алфавита в соответствии с машинными кодами не идентична их алфавитной последовательности.

Оператор условного перехода позволяет использовать значения, вводимые пользователем, для управления ходом выполнения программы.

Пример 4.16

```
:10  INPUT  "ЧИСЛА",A,B
:20  INPUT  "СЛОЖИТЬ (С), УМНОЖИТЬ (У)",A⊙
:30  IF  A⊙="С"  THEN  60
:40  IF  A⊙="У"  THEN  80
:50  GOTO  20
:60  PRINT  "СУММА=";A+B
:70  GOTO  90
:80  PRINT  "ПРОИЗВЕДЕНИЕ=";A*B
:90  PRINT "ДЕЙСТВИЕ ВЫПОЛНЕНО"
```

В строке 10 записан оператор ввода значений двух числовых переменных А и В с выдачей в качестве подсказки строки «ЧИСЛА»; в строке 20 —оператор ввода значений символьной переменной А⊙, при этом пользователю подсказывается, что он должен ввести символ С, если нужно, чтобы программа выполнила сложение, или символ У — если умножение.

В строке 30 оператор условного перехода проверяет, является ли значением А⊙ символ С. Если это условие выполняется, то осуществляется переход к строке 60 с оператором печати строки «СУММА = » и суммы значений числовых переменных А и В. После этого по оператору безусловного перехода в строке 70 осуществляется переход к оператору печати сообщения о выполнении действия (строка 90).

Если условие оператора в строке 30 не выполняется, то будет выполнен следующий по порядку оператор — оператор условного перехода в строке 40. Этот оператор проверяет, является ли значением А⊙ символ У. Если это условие выполняется, осуществляется переход к строке 80 с оператором печати строки «ПРОИЗВЕДЕНИЕ = » и произведения значений числовых переменных А и В. После этого по оператору PRINT в строке 90 печатается сообщение о выполнении действия.

Если условия в строках 30 и 40 не выполняются, т. е. значение символьной переменной А⊙ не равно С или У, то следующим выполняется оператор GOTO в строке 50. В этом случае пользователю повторно будет предложено выбрать операцию сложения или умножения введенных ранее чисел. Ниже приведен примерный диалог при выполнении этой программы.

```
:RUN
ЧИСЛА? 20,8
СЛОЖИТЬ (С), УМНОЖИТЬ (У)? У
ПРОИЗВЕДЕНИЕ=160
ДЕЙСТВИЕ ВЫПОЛНЕНО
:RUN
ЧИСЛА? 96,15.2
СЛОЖИТЬ (С), УМНОЖИТЬ (У)? +
СЛОЖИТЬ (С), УМНОЖИТЬ (У)? С
СУММА=111.2
ДЕЙСТВИЕ ВЫПОЛНЕНО
:_
```

В одном операторе IF — THEN могут непосредственно задаваться сразу несколько отношений, объединенных логическими связками **AND** (И), OR (ИЛИ) и XOR (ИСКЛЮЧАЮЩЕЕ ИЛИ), например,

```
IF A>B*2 AND R⊙ = "НЕТ" THEN 200
IF H⊙="18.11.11" OR Z=100 AND K<0 THEN 1000
```

Если два отношения объединены связкой AND, то условие выполняется, когда выполняются оба

отношения. Если два отношения объединены связкой OR, то условие выполняется, когда выполняется хотя бы одно из отношений. Если два отношения объединены связкой XOR, то условие выполняется, когда выполняется одно из двух отношений, но не оба одновременно. Таким образом, рассмотренные четыре условия

```
200  IF A+B=0 THEN 1000
210  IF Z=14*K THEN 1000
220  IF C=0 THEN 1000
230  IF S=2 THEN 1000
```

могут быть записаны в одном операторе

```
200  IF  A+B=0  OR   Z=14*K  OR  C=0  OR   S = 2  THEN   1000
```

Заметим, что если в условии связывается более двух отношений, то проверка осуществляется последовательно слева направо. Сначала проверяется крайняя слева пара отношений.

```
A+B=0 OR Z = I4*K
```

Затем проверяется результат этой проверки и соседнее справа отношение, т. е. как бы

```
(A+B=0  OR  Z = 14*K)   OR  C = 0
```

После этого результат проверки сравнивается со следующим отношением

```
((A + B = 0  OR  Z = 14*K)   OR  C = 0) OR  S = 2
```

и т. д., пока не будут проверены все отношения.

4.6.Комментарии в тексте программы

Чтобы облегчить последующую работу с текстом программы, при программировании в нее целесообразно включать поясняющий текст, не влияющий на исполнение программы. В Бейсике для этого служит оператор REM. За словом REM в программной строке может следовать любая последовательность символов Бейсика, кроме, конечно, двоеточия, используемого в качестве разделителя между операторами в программной строке. Например,

```
10  REM  ПРОГРАММА  РАСЧЕТА  КОРРЕЛЯЦИИ
```

При выполнении программы оператор REM игнорируется.

Включение комментариев в текст программы является весьма эффективным средством ее документирования. Чем более подробны и наглядны комментарии, тем проще последующая работа с программным текстом. Для ускорения выполнения программы и сокращения занимаемого ею объема памяти комментарии могут быть исключены при формировании рабочего варианта программы. В связи с этим целесообразно в строки с оператором REM не включать никаких других операторов, а также не ссылаться на номера этих строк в операторах переходов.

4.7. Математические функции

В качестве числовых данных, помимо числовых констант и числовых переменных, могут использоваться элементарные математические функции, аргументами и значениями которых являются числа. Математические функции, включенные в Бейсик «Искры 226», и пояснение к ним приводятся ниже.

Функция		Пояснение
S1N	(‹аргумент›)	Синус аргумента
COS	(‹аргумент›)	Косинус аргумента
TAN	(‹аргумент›)	Тангенс аргумента
ARCSIN	(‹аргумент›)	Арксинус аргумента
ARCCOS	(‹аргумент›)	Арккосинус аргумента
ARCTAN	(‹аргумент›)	Арктангенс аргумента
RND	(‹аргумент›)	Случайное число между 0 и 1
ABS	(‹аргумент›)	Абсолютное значение аргумента
INT	(‹аргумент›)	Наибольшее целое число, не превосходящее аргумент
SGN	(‹аргумент›)	1 при положительном аргументе, 0 при нулевом и — 1 при отрицательном
LOG	(‹аргумент›)	Натуральный логарифм аргумента
EXP	(‹аргумент›)	Число е в степени аргумент

SQR	(‹аргумент›)	Квадратный корень аргумента
ROUND	(‹аргумент 1›, ‹аргумент 2›)	Округление аргумента 1 до разряда, задаваемого целой частью аргумента 2
#P1		Число π (т. е. 3.14159265359)

При записи функции указывается имя функции и один или несколько аргументов в скобках. Если аргументов несколько, они перечисляются через запятую. Например,

```
COS  (.125+A)
LOG (3000)
ROUND (TAN(2.18)*2,3)
```

Остановимся на некоторых особенностях использования функций.

После включения машины или выполнения оператора CLEAR аргументы тригонометрических функций воспринимаются в радианах. Если аргумент задается в градах ($2\pi = 400$ град), предварительно должен быть выполнен оператор SELECT G, а в случае, если используется единица измерения градусы, — оператор SELECT D. Для возврата к вычислениям в радианах следует выполнить оператор SELECT R.

Пример 4.17

Следующая программа позволяет вычислять значение синуса при задании угла в градусах с последующим возвращением к заданию угла в радианах.

```
:10 REM ВЫЧИСЛЕНИЕ СИНУСА УГЛА
:20 INPUT "УГОЛ В ГРАДУСАХ",U
:30 SELECT D
:40 PRINT SIN(U)
:50 SELECT R
:RUN
УГОЛ В ГРАДУСАХ?90
  1
:__
```

Функция RND позволяет генерировать псевдослучайные числа от 0 до 1. Единственным значением аргумента, влияющим на генерацию чисел, является 0. Функция RND с нулевым аргументом обеспечивает генерацию повторяющихся последовательностей случайных чисел, что полезно, например, при отладке программ с этой функцией.

Пример 4.18

```
:10 PRINT RND(0)
:20 PRINT RND(0)
:30 PRINT RND(0)
:RUN
.9250598142006
.7496318006595
.7152565785556
:RUN
.9250598142006
.7496318006595
.7152565785556
```

Теперь изменим в программе аргумент функции RND, например на 1, и снова выполним ее дважды.

```
:10 PRINT RND(l)
:20 PRINT RND(l)
:30  PRINT  RND(l)
:RUN
.1597716054051
.7865258535282
.0092828197479
:RUN
.4905779638641
.2827368719862
.8830627096022
:__
```

Функция INT вычисляет в качестве значения ближайшее меньшее целое число, например

```
PRINT  INT(256.9)
256
PRINT  INT(—17.2)
– 18
:—
```

Пример 4.19

```
:10    REM ОПРЕДЕЛЕНИЕ ВИСОКОСНОГО ГОДА
:20    INPUT "ГОД",A
:30    PRINT A;"-  ";
:40    IF INT(A/4)=A/4  THEN  50
:50    PRINT "НЕ ";
:60    PRINT "ВИСОКОСНЫЙ ГОД"
```

Программа запрашивает год и печатает введенное числовое значение и тире с пробелом после него. Затем целая часть числа, полученная от деления А на 4, сравнивается с частным. Если они равны, т. е. А делится на 4 без остатка, то печатается сообщение о том, что год високосный, в противном случае печатается сообщение о том, что год не високосный.

```
:RUN
ГОД?  1986
1986 — НЕ ВИСОКОСНЫЙ ГОД
:RUN
ГОД?  1992
1992 —ВИСОКОСНЫЙ ГОД
:—
```

Теперь рассмотрим два примера возможного использования функции ABS, вычисляющей абсолютное значение аргумента. Следующая последовательность операторов условного перехода, задающая переход к строке 130, при одновременном выполнении условий

```
100  IF  A+5=0  THEN  110:GOTO  1000
110  IF  B = 0  THEN  120: GOTO  1000
120  IF  K=5*B THEN  130:GOTO  1000
```

равносильна

```
100 IF ABS(A+5)+ABS(B)+ABS(K—5*B)=0 THEN 130: GOTO 1000
```

С помощью функции ABS можно также сразу проверить, находится ли значение арифметического выражения в заданном интервале. В частности, последовательность из двух операторов условного перехода

```
100  IF A<1 THEN  110:GOTO  1000
110  IF A>0 THEN  120:GOTO  1000
```

равносильна оператору

```
100  IF  ABS(2*A—1)<1  THEN  120:GOTO  1000
```

Функция ROUND используется для округления числовых данных

ROUND (<округляемое значение>, <округляемый разряд>)

Значением функции является значение первого аргумента, округленное до разряда, заданного целой частью второго аргумента.

Если второй аргумент больше 0, то округление осуществляется до соответствующего разряда после десятичной точки, например

```
:PRINT ROUND(1.23,l)
1.2
:—
```

Если второй аргумент меньше или равен 0, то округление осуществляется до следующего по порядку разряда до десятичной точки (считая слева направо). Например,

```
: PRINT  ROUND  (—3.8,0)
—4
: PRINT  ROUND(1256,–2)
: PRINT  ROUND (1256,–2)
1300
:—
```

4.8. Определение числовых функций пользователем

В языке Бейсик есть возможность наряду с имеющимися в языке элементарными функциями использовать новые функции, включая определения этих функций непосредственно в текст программы. В качестве новых функций могут использоваться любые арифметические выражения, включающие числа, переменные и элементарные функции, а также другие функции, определенные в данной программе.

Определение новых функций осуществляется с помощью оператора DEFFN. Оператор DEFFN имеет следующий синтаксис

DEFFN <имя функции> (<формальная переменная>) = <а.в.>

В качестве имени функции может использоваться любая латинская буква от A до Z или любая цифра от 0 до 9. Формальная переменная выступает в качестве аргумента определяемой функции, значение ее должно задаваться при обращении к ней. Именем формальной переменной может быть имя любой числовой переменной Бейсика. Функция может быть объявлена в любом месте программы, независимо от того, где она будет использоваться. Например, оператор объявления функции может записываться

```
DEFFN  B(R)=R + SIN(R)
```

При использовании функции, определенной подобным образом, указывается слово FN, за ним имя функции и арифметическое выражение в скобках

FN<имя функции> (а.в.),

например

```
FN  B(P + 25*D2)
FN  2(4852)
```

При использовании такой функции сначала вычисляется значение арифметического выражения, затем это значение присваивается формальной переменной и осуществляется вычисление значения выражения, заданного в определении функции.

Пример 4.20

```
:10    REM ПРИМЕР  ОПРЕДЕЛЕНИЯ ФУНКЦИИ В ПРОГРАММЕ
:20    DEFFN  A(H) = (H/\2– A)/\(1/3)
:30    A=56
:40    PRINT  FN  A(8), FN  A(9),   FN  A(10)
: RUN
  2                    2.924017738212     3. 530348335325
:_
```

4.9. Операторы задания констант

Как уже говорилось, постоянные значения могут быть заданы в программах и присвоены переменным с помощью оператора LET. Для задания большого числа постоянных значений удобно использовать операторы Бейсика DATA, READ и RESTORE.

Значения, которые должны быть присвоены переменным, перечисляются через запятую в операторе DATA. Например,

```
DATA  1986, 1990,  "ПЛАН", "ОТЧЕТ"
```

В операторе DATA допускается также указывать имена символьных переменных, значения которых играют роль соответствующих констант, например

```
DATA  1986,E,M⊙,"ОТЧЕТ"
```

Переменные, которым нужно присвоить значения, содержащиеся в операторе DATA, в том же порядке перечисляются в операторе READ. Например,

```
READ  A,B,X⊙,У⊙
```

При выполнении оператора READ отыскивается оператор DATA, независимо от того, в каком месте программы он находится, и значения, перечисленные в DATA, присваиваются переменным в операторе READ. Таким образом, пара операторов

```
DATA  1986, 1990,  "ПЛАН", "ОТЧЕТ"
READ  A,B,X⊙,Y⊙
```

равносильна последовательности операторов

```
A=1986
B = 1990
X⊙ = "ПЛАН"
Y⊙ = "ОТЧЕТ"
```

Константы могут перечисляться в нескольких операторах DATA, расположенных в произвольных местах программы. Например, вместо

```
DATA   1986,1990,  "ПЛАН", "ОТЧЕТ"
```

можно записать

```
DATA   1986,1990
DATA  "ПЛАН","ОТЧЕТ"
```

или, скажем

```
DATA   1986,1990,"ПЛАН"
DATA  "ОТЧЕТ"
```

Переменные могут перечисляться в нескольких операторах READ, важно только соблюдать порядок следования переменных в этих операторах: каждой следующей переменной присваивается следующее по порядку значение из операторов DATA. В частности,

```
READ  A,B,X⊙,Y⊙
```

равносильно

```
 READ A,B
 READ  X⊙,Y⊙
```

или

```
READ   A
READ   B,X⊙,Y⊙
```

Пример 4.21

Рассмотрим программу, позволяющую вычислить порядковый номер дня в невисокосном году.

```
:10   REM ВЫЧИСЛЕНИЕ НОМЕРА ДНЯ
:20   DATA "ЯНВАРЬ", 31, "ФЕВРАЛЬ", 28, "МАРТ", 31
:30   DATA "АПРЕЛЬ", 30, "МАЙ", 31, "ИЮНЬ", 30
:40   DATA "ИЮЛЬ", 31."АВГУСТ", 31, "СЕНТЯБРЬ", 30
:50   DATA "ОКТЯБРЬ", 31 ,"НОЯБРЬ", 30 ,"ДЕКАБРЬ", 31
:60   INPUT "ЧИСЛО", D
:70   INPUT "МЕСЯЦ",M⊙
:80   READ M1⊙. Dl
:90   IF M⊙=M1⊙ THEN 120
:100  N=N+D1
:110  GOTO 80
:120  PRINT  N+D;"-Й  ДЕНЬ  ГОДА"
:—
```

В строках 20—50 заданы константы — наименование каждого месяца и количество дней в нем; в строках 60 и 70 — операторы ввода с клавиатуры номера дня и наименования месяца.

По оператору READ в строке 80 считывается наименование первого месяца и количество дней в нем. Если название этого месяца совпадает с введенным (строка 90), то печатается ответ — сумма переменной N (при запуске обнуляется) и введенного числа дней (строка 120). Если название не совпадает, то число дней первого месяца добавляется к переменной N (строка 100) и считывается следующее число. Этот процесс считывания наименований месяцев и суммирования числа дней в них будет продолжаться до тех пор, пока считанное из оператора DATA наименование месяца не совпадет с введенным наименованием. В качестве ответа будет напечатана накопленная в N сумма дней предыдущих месяцев плюс введенное число.

В случае, если пользователь неправильно набрал наименование месяца, ни одно из 12 считанных наименований месяца не совпадет с введенным. При попытке считывания 13-й пары данных в операторе система выдаст сообщение об ошибке (ERR 27), поскольку в операторах DATA нет больше констант для оператора READ.

Выполним эту программу.

```
:RUN
ЧИСЛО?   15
```

```
МЕСЯЦ?  МАРТ
74-й  ДЕНЬ  ГОДА
:__
```

Не всегда удобно осуществлять последовательное считывание данных, заданных в операторах DATA. Используя оператор RESTORE, можно осуществлять повторное считывание, начиная с любой по порядку константы оператора DATA.

Оператор RESTORE без параметров устанавливает специальный указатель начала считывания данных на первую константу первого оператора DATA в программе. В результате очередной оператор READ начнет считывать данные, начиная с начала.

При выполнении оператора RESTORE в форме

```
RESTORE<а.в.>
```

вычисляется целая часть значения арифметического выражения и указатель устанавливается на соответствующую по порядку константу операторов DATA программы. Очередной оператор RESTORE начнет считывания с этой константы. Значение арифметического выражения должно быть в пределах от 1 до 9999. Рассмотрим пример.

П р и м е р 4.22

```
:10   REM ПЕРЕВОД НОМЕРА МЕСЯЦА В НАИМЕНОВАНИЕ
:20   INPUT "НОМЕР МЕСЯЦА", M
:30   RESTORE M
:40   READ M⊙
:50   PRINT M⊙
:60   DATA "ЯНВАРЬ",  "ФЕВРАЛЬ",  "МАРТ",  "АПРЕЛЬ",
"МАЙ",  "ИЮНЬ", "ИЮЛЬ",  "АВГУСТ",  "СЕНТЯБРЬ",
"ОКТЯБРЬ",  "НОЯБРЬ",  "ДЕКАБРЬ"
```

При выполнении оператора RESTORE введенный номер месяца используется для установки указателя начала считывания на наименование соответствующего месяца в операторе DATA.

```
:RUN
НОМЕР  МЕСЯЦА?11
ДЕКАБРЬ
:__
```

При выполнении оператора RESTORE в форме с номером строки

```
RESTORE,< номер строки>
```

при установке указателя отсчет начинается с первой константы DATA указанной строки. Например, по оператору

```
RESTORE,120
```

указатель будет установлен на первой константе оператора DATA в строке 120. Если при этом задается и арифметическое выражение

```
RESTORE<а.в.>, <номер строки>
```

то указатель устанавливается на соответствующий по порядку элемент оператора DATA. Например, по оператору

```
RESTORE 4, 120
```

указатель установится на четвертой константе оператора DATA в строке 120.

Глава 5. ЗАПИСЬ И ЗАГРУЗКА ПРОГРАММ НА МАГНИТНЫЕ ДИСКИ

5.1. Режим автоматической каталогизации файлов

Обращение к магнитному диску может осуществляться в двух режимах: в режиме абсолютной адресации секторов и в режиме автоматической каталогизации файлов (под файлами понимаются программы или совокупности данных, записанные на диск).

В режиме абсолютной адресации секторов при записи или считывании информации пользователь должен задавать непосредственно физические номера секторов. В режиме автоматической каталогизации файлов адреса секторов определяются самой системой. На диске создается так

называемый каталог, состоящий из указателя (т. е. оглавления) каталога и области собственно каталога.

В настоящей главе рассматривается запись и загрузка программ в режиме автоматической каталогизации файлов. При записи программы пользователь указывает только имя создаваемого программного файла, а машина записывает программу из памяти в свободные секторы области каталога и заносит имя файла и номера начального и конечного секторов файла в указатель каталога. При загрузке (т. е. считывании с диска и помещении в оперативную память) программы пользователь опять-таки указывает лишь имя программного файла. Машина, используя сведения о размещении файла, хранящиеся в указателе каталога, отыскивает программу в области каталога и загружает ее в память.

Прежде чем работать с диском в режиме автоматической каталогизации файлов, необходимо организовать на нем каталог. Для этого предназначен оператор SCRATCH DISK, в котором указываются число секторов в указателе каталога и номер последнего сектора, входящего в область каталога. В случае, если число секторов в указателе каталога не задано, оно устанавливается равным 24. Оператор SCRATCH DISK имеет следующий синтаксис[1]:

SCRATCH DISK $\left\{\begin{matrix}F\\ \end{matrix}\right\}$ [LS=<а.в.>,] END=<а.в.>

Целая часть первого арифметического выражения задает число секторов, которые должны быть отведены под указатель каталога, а целая часть второго — номер последнего сектора каталога. Указатель каталога всегда начинается с нулевого сектора диска, после указателя идет область каталога. Например, оператор

```
:SCRATCH  DISK  F  LS = 6,  END = 900
:_
```

формирует каталог на диске F (т. е. правом, верхнем или фиксированном, в зависимости от типа накопителя), причем под указатель отводится 6 секторов (т. е. с 0-го по 5-й сектор), а область каталога кончается в 900-м секторе (т. е. область каталога размещается с 6-го по 900-й секторы).

Каталог можно формировать как на новом диске, прошедшем предварительно процедуру форматирования, так и на уже использовавшемся диске. В последнем случае информация, ранее записанная на диск, будет игнорироваться (в пределах каталога).

Число секторов, отводимых под указатель каталога, в последующем изменено быть не может. Число секторов, которое следует зарезервировать под указатель, зависит от максимального числа файлов, которые будут занесены в каталог. В 0-м секторе может быть размещено до 15 имен файлов, во всех остальных — до 16. Под указатель допускается отводить от 1 до 255 секторов.

Файлы размещаются в секторах области каталога друг за другом в порядке их создания.

Примечание. Последовательность имен файлов в указателе может не соответствовать последовательности размещения соответствующих файлов в области каталога: номер сектора указателя каталога для хранения имени файла выбирается в соответствии с некоторой функцией, зависящей от имени файла, что в последующем ускоряет поиск информации о файле.

Область каталога может быть увеличена в процессе работы вплоть до последнего сектора диска с помощью оператора MOVE END, в котором указывается, в какой сектор диска следует переместить конец каталога

MOVE END $\left\{\begin{matrix}F\\R\end{matrix}\right\}$ =<а.в.>

Целая часть арифметического выражения задает номер сектора, в который должен переместиться конец каталога (номер этого сектора не должен быть меньше номера сектора, являющегося текущим концом каталога). Например, с помощью оператора

```
:MOVE  END  F = 918
:_
```

конец каталога будет перенесен в сектор 918.

Бейсик микроЭВМ «Искра 226» включает специальный оператор для выдачи информации, содержащейся в указателе каталога. По оператору LIST DC на устройство вывода на экран выдается следующая информация о каталоге в целом:

тип диска, на котором находится каталог (FIXED для диска F и REMOVABLE для диска R);

[1] Здесь и далее конструкции, заключенные в квадратные скобки, не являются обязательными, а из параметров, заключенных в фигурные скобки, может использоваться любой параметр.

число секторов в указателе каталога (INDEX SECTORS);
номер последнего сектора, отведенного под каталог (END CAT. AREA);
номер сектора, являющегося текущим концом каталога (CURRENT END).
Далее следует перечень файлов, содержащихся в каталоге, с указанием для каждого файла:
имени файла (NAME);
типа файла (TYPE): P — программный, D — файл данных,
SP — исключенный (вычеркнутый) из каталога программный файл,
SD — исключенный (вычеркнутый) из каталога файл данных;
номера сектора, с которого начинается файл (START);
номера последнего сектора файла (END);
числа использованных секторов, т. е. секторов, реально занятых в файле (USED).

Если после создания каталога с помощью оператора SCRATCH DISK на диск не записывались программы или данные, то в результате выполнения оператора LIST DC будет выдана только информация о каталоге в целом.

```
:LIST DC F
FIXED CATALOG
INDEX SECTORS=00006
END CAT.AREA=00910
CURRENT END=00005

NAME    TYPE  START  END   USED
```

5.2. Запись и загрузка программ

Для записи в каталог программы, находящейся в данный момент в оперативной памяти микроЭВМ, используется оператор SAVE DC. В операторе SAVE DC в форме символьной константы или символьной переменной задается имя, под которым программа должна быть занесена в каталог. Имя может состоять из любых символов, в том числе символов русского алфавита, его длина не должна превышать 8 символов.

Пример 5.1

Введем в оперативную память мини-ЭВМ следующую программу:

```
:10  REM  РАСЧЕТ  ПЛОЩАДИ  И  ОБЪЕМА
:20  INPUT  "ДЛИНА", A
:30  INPUT  "ШИРИНА", B
:40  INPUT  "ВЫСОТА", C
:50  PRINT  "ПЛОЩАДЬ=";A*B, "0БЪЕМ=";A*B*C
```

Запишем эту программу в каталог на диске F под именем «ПРОГ1»

```
:SAVE  DC  F  "ПРОГ1"
:_
```

В результате операции в каталог будет включен новый программный файл. Предполагается, что ранее в каталоге файла с таким именем не было, иначе записи не произойдет и будет выдано сообщение об ошибке.

Если теперь выполнить оператор LIST DC, то мы увидим, что в указателе каталога появилось имя «ПРОГ1», метка типа файла R (программный), номера секторов диска, связанных с этим программным файлом (START — начальный, END — конечный, USED — число использованных)

```
:LIST DC F
FIXED CATALOG
INDEX SECTORS=00006
END CAT.AREA=00918
CURRENT END=00008

NAME   TYPE    START   END    USED
ПРОГ1  P       00006    00008  00003
```

С помощью оператора SAVE DC на диск может быть записана не вся программа, а только ее часть. Для этого в операторе SAVE DC после имени программного файла следует указать номер начальной и через запятую номер конечной строки записываемого фрагмента, например

```
SAVE  DCR  "ТАБЛ" 20,800
```

Если опущены начальный

```
SAVE  DCR  "ТАБЛ",  800
```

или конечный номера строк

```
SAVE  DCR  "ТАБЛ".  20
```

программа записывается соответственно с начала или до конца. Запишем строки указанной выше программы, начиная с 20-й строки, на диск F под именем «ПРОГ2»

```
SAVE   DCF  "ПРОГ2"  20
:_
```

В операторе SAVE DC после параметра F или R, задающего, на какой из дисков дисковода должна осуществляться запись, может следовать символ ⊙, например

```
SAVE   DCR   ⊙   "РАСЧЕТ"
```

При наличии этого параметра после записи программы автоматически осуществляется контрольное считывание. Это несколько увеличивает время выполнения операции, но позволяет контролировать правильность записи программы на магнитный диск.

В оператор SAVE DC может также включаться параметр, задающий специальные форматы записи программы

$$\left\{\begin{matrix} T \\ R \\ G \end{matrix}\right\}$$

например

```
SAVE DCR T "СТАРТ"
SAVE DCR ⊙P "ПОИСК"
SAVE DCR G "ПРОГ"
```

Если специфицирован параметр T, программа записывается на диск в оттранслированной форме, т. е. в некотором внутреннем формате машины, что сокращает место, занимаемое программой на диске, и время обработки программных строк при загрузке программы с диска.

Программа, записанная с помощью оператора SAVE DC с параметром P, защищена от просмотра и записи на диск. При загрузке такой защищенной программы в оперативную память предусматривается только ее выполнение. При этом режим защиты оперативной памяти от просмотра программного текста сохраняется до тех пор, пока не будет выполнен оператор CLEAR.

Параметр G задает запись программы на диск одновременно в оттранслированной и защищенной форме.

Для того чтобы загрузить в память программу, хранящуюся в каталоге диска, используется оператор LOAD DC, в котором указывается имя загружаемой программы. Не следует забывать перед загрузкой программы очищать оперативную память с помощью оператора CLEAR.

Пример 5.2

Чтобы загрузить записанную нами программу «ПРОГ1» с диска F, следует набрать

```
:LOAD  DC   F   «ПРОГ1»
:_
```

Теперь все строки программы «ПРОГ1» считаны с диска и загружены в оперативную память. При желании это можно проверить, выполнив оператор вывода текста программы

```
:LIST
10   REM РАСЧЕТ ПЛОЩАДИ И ОБЪЕМА
20   INPUT "ДЛИНА",A
30   INPUT "ШИРИНА",B
40   INPUT "ВЫСОТА",C
50   PRINT "ПЛОЩАДЬ=";A*B,"ОБЪЕМ = ";A*B*C
```

Если перед выполнением оператора загрузки программы LOAD DC в оперативной памяти оставался программный текст, то произойдет как бы «наложение» новой программы на старую. Строки старой программы, имеющие одинаковые номера с номерами строк новой программы, будут изменены на новые, а остальные останутся прежними.

Наложение можно использовать для компоновки программы из отдельных сегментов, записанных на магнитный диск. Оператор LOAD DC с заданием начального и конечного номеров строк обеспечивает стирание сегмента старой программы перед загрузкой новой

```
LOAD  DC  R  "МОДУЛЬ  2"  30,80
```

Если опущены начальный номер строки

```
LOAD  DC  R  "МОДУЛЬ 2",  80
```

или конечный номер строки

```
LOAD  DC  R  "МОДУЛЬ 2" 30
```

стирается сегмент программы соответственно с начала или до конца программы.

Если же в операторе указан третий номер программной строки, то после стирания и загрузки инициируется выполнение программы, находящейся в памяти, начиная с этого номера. Например, но оператору

```
LOAD  DC  R  "МОДУЛЬ 2"  30,80,10
```

сотрется сегмент программы в оперативной памяти с 30-й по 80-ю строки, затем загрузится программа под именем «МОДУЛЬ 2» и после этого начнется выполнение получившейся программы, начиная со строки 10. Удобно пользоваться этой формой оператора для загрузки и запуска программы, записанной на диске. Для этого достаточно набрать

```
LOAD  DC  R "МОДУЛЬ 2"  „ 0
```

Этот оператор равнозначен последовательности операторов

```
CLEAR
LOAD DC  R "МОДУЛЬ  2"
RUN
```

Заметим, что все, сказанное об операторе LOAD DC, относится к выполнению этого оператора в режиме непосредственного счета, т. е. когда он вводится после сообщения «:». Особенности выполнения оператора LOAD DC в программном режиме описаны в гл. 19.

Если файл, ранее занесенный в каталог, устарел и не нужен пользователю, в указатель заносится специальная пометка о его исключении. На место исключенного файла может быть записан любой другой файл (в пределах секторов, которые были отведены под исключенный файл в каталоге).

Пометка ненужных файлов в каталоге осуществляется с помощью оператора SCRATCH путем указания в кавычках имени файла (не следует путать оператор SCRATCH с ранее рассмотренным оператором формирования каталога SCRATCH DISK). Например, оператор

```
:SCRATCH  F  "ПРОГ1".
:__
```

помечает в указателе каталога файл по имени «ПРОГ1» (на диске F) как исключенный. При просмотре указателя каталога с помощью оператора LIST DC вычеркнутые файлы индицируются буквой S, которая приписывается к букве, указывающей на прежний тип файла. Исключенный программный файл, в частности, отмечается в указателе как SP.

```
:L1ST DC F
FIXED CATALOG
INDEX SECTORS=00006
END CAT.AREA=00918
CURRENT END=0001l

NAME   TYPE    START     END     USED
ПРОГ1  SP      00006     00008   00003
ПР0Г2  P       00009     00011   00003
```

Если нужно сразу исключить несколько файлов, то в операторе SCRATCH их имена могут перечисляться через запятую

```
SCRATCH  F  "PROGRAM",  "INIT",  "МАССИВ"
```

Загрузка вычеркнутой программы в память микроЭВМ в режиме автоматической каталогизации файлов невозможна, однако она может быть осуществлена в режиме абсолютной адресации секторов (см. 18.9.2).

Для записи программы на место, которое занимает файл, помеченный как исключенный, используется модификация оператора записи SAVE DC.

SAVE DC {F / R} "<новое имя>" ("<старое имя>")

Пример 5.3

Для записи программы

```
:10 REM ВЫЧИСЛЕНИЕ ПЛОЩАДИ КРУГА
:20 INPUT "РАДИУС",R
:30 PRINT "ПЛОЩАДЬ=";≠PI*P/\2
```

под именем «КРУГ» на место, занимаемое ранее программой «ПРОГ1», следует выполнить

```
:SAVE DC ("ПРОГ1") "КРУГ"
:__
```

Запись прошла успешно, при этом обновилась и информация в указателе каталога.

```
:LIST DC F
FIXED CATALOG
INDEX SECTORS=00006
END CAT.AREA=00918
CURRENT END=00011
NAME   TYPE   START    END    USED
КРУГ   P      00006    00008  00003
ПРОГ2  P      00009    00011  00003
```

Если программа, записываемая на место файла, помеченного как исключенный, не помещается на нем полностью, выдается соответствующее сообщение об ошибке, а информация в указателе каталога не изменяется. Пользователь может записать не поместившуюся программу на место какого-либо другого исключенного файла большего размера или же записать ее как новый программный файл.

При первоначальной записи программного файла в каталог можно заранее зарезервировать некоторое число секторов дополнительно к тем секторам, которые займет записываемая программа. Это позволит в последующем на то же место записывать, например, новые варианты программы, имеющие большую длину, чем первоначальный вариант. Для резервирования секторов используется следующая модификация оператора записи программы SAVE DC:

SAVE DC {F / R} (<а.в.>) " <имя>"

где целая часть арифметического выражения задает число секторов, которые при записи будут добавлены в файл в качестве резерва.

Например, по оператору

```
SAVE DC F(5) "РАСЧЕТ"
```

в каталог под именем «РАСЧЕТ» записывается программа из оперативной памяти, причем под файл отводится место, необходимое для записи этой программы, плюс пять резервных секторов.

Со временем в каталоге может накопиться большое число файлов, помеченных как исключенные, которые по различным причинам не используются для записи информации. Это приводит к неэффективному использованию каталога, и следует воспользоваться оператором MOVE для реорганизации каталога. По этому оператору из каталога исходного диска в каталог, предварительно организованный на другом диске с помощью оператора SCRATCH DISK, последовательно копируются все файлы, за исключением вычеркнутых.

В операторе MOVE за словом MOVE указывается исходный диск, а после слова TO — диск, на который должны скопироваться файлы. Например, оператор

```
MOVE F TO R
```

переносит невычеркнутые файлы из каталога диска F в каталог диска R.

С помощью оператора MOVE можно также копировать каталогизированные файлы с диска на диск. Для этого в операторе необходимо указать имя копируемого файла. Например, по оператору

```
MOVE F "START" TO R
```

в каталог диска R будет занесена копия файла «START» из каталога диска F,

При копировании файла в новом каталоге при необходимости можно зарезервировать для него дополнительное число секторов, например

```
MOVE  F «START» TO  R(8)
```

Файл можно скопировать на место, занимаемое вычеркнутым файлом, например по оператору

```
MOVE  F  "START"  TO  R("ПРОГ1")
```

файл «START» из каталога диска F будет скопирован на место вычеркнутого файла «ПРОГ1» в каталоге диска R.

Число программ, содержащихся в каталоге, может быть весьма велико, что затрудняет просмотр указателя каталога с помощью оператора LIST. В этом случае можно воспользоваться оператором LIST с маской, обеспечивающим выборочный **просмотр** указателя.

Маска представляет собой строку символов, которая в операторе LIST задается символьной константой или символьной переменной, следующей после спецификации диска. При выполнении оператора LIST с маской каждый символ имени файла сравнивается с соответствующим по порядку символом маски. Выводятся сведения только о тех файлах, в имени которых в каждой позиции стоит тот же символ, что и в маске, за исключением тех позиций, в которых в маске стоят пробелы. Например, оператор

```
LIST  DC  F  "У"
```

выводит только те имена файлов, у которых третьим символом является У

```
:LIST DC F "  У"
FIXED CATALOG
INDEX SECTORS=00006
END CAT.AREA=00918
CURRENT END=00011

NAME    TYPE   START  END    USED
КРУГ    P      00006  00008  00003
```

Глава 6. АДРЕСАЦИЯ УСТРОЙСТВ И ИСПОЛЬЗОВАНИЕ ПЕЧАТАЮЩЕГО УСТРОЙСТВА

6.1. Оператор адресации устройств

Одни и те же операции, связанные с вводом и выводом данных, могут осуществляться на различных устройствах ввода-вывода ЭВМ. Например, значения переменных могут выводиться на экран или печатающее устройство, запись программ и данных может производиться на гибкие или на жесткие магнитные диски и т. д. Каждое устройство, подключенное к ЭВМ, имеет определенный адрес, который используется для адресации операций ввода-вывода. Этот адрес кодируется двухразрядным шестнадцатеричным числом (в качестве цифр от 0 до 15 используются цифры от 0 до 9 и далее латинские буквы от A до F), называемым физическим адресом устройства. Физический адрес, например экрана, — 05, клавиатуры — 01, накопителя на гибком магнитном диске—18, накопителя на жестком магнитном диске—1C, алфавитно-цифрового печатающего устройства (АЦПУ) —0C.

По умолчанию для каждой операции ввода-вывода система автоматически выбирает устройство с определенным адресом. Ниже приводятся адреса устройств, выбираемые автоматически.

Группа операций ввода-вывода	**Физические адреса устройств**
CI (консольный ввод, т. е. ввод при диалоге с системой)	01 (консольная клавиатура)
CO (консольный вывод, т. е. вывод при диалоге **с** системой)	05 (консольный экран)
LIST (вывод по операторам LIST, LIST DC)	05 (консольный экран)
PRINT (вывод по оператору PRINT)	05 (консольный экран)
INPUT (ввод по операторам INPUT и LINPUT)	01(консольная клавиатура)
DISK (ввод-вывод на магнитный диск)	18F (накопитель на гибких магнитных дисках, правый (нижний) диск — F)

Если же необходимо адресовать ту или иную операцию на устройство с другим адресом, то это можно сделать двумя способами.

Во-первых, можно непосредственно адресовать каждый конкретный оператор ввода-вывода. Для этого в операторе, как правило, сразу после ключевого слова, указывается физический адрес требуемого

устройства (адреса устройств приведены в приложении 2) в следующей форме

/<физический адрес устройства>,

Например, по оператору

```
SAVE DC F/1C,  "ПРОГ1"
```

запись программы в режиме автоматической каталогизации файлов будет выполняться на устройство с адресом 1С — накопитель на жестких магнитных дисках (по параметру F выбирается фиксированный диск).

Во-вторых, можно изменить адрес для, любой группы операций ввода-вывода. Для этого используется оператор выбора адреса SELECT. После слова SELECT указывается наименование группы операций и адрес устройства, которое должно назначаться по умолчанию для соответствующих операций. Например, чтобы тот же диск с адресом 1CF выбирался автоматически для дисковых' операций следует указать

```
SELECT DISK  1CF
```

В операторе SELECT допустимо перечисление, например

```
SELECT  DISK  1CF,  PRINT  OC
```

Выбор устройства, сделанный с помощью оператора SELECT, справедлив до тех пор, пока не будет выполнен следующий оператор SELECT с назначением другого устройства для той же операции или оператор CLEAR.

6.2. Работа с АЦПУ

Чтобы управлять алфавитно-цифровым печатающим устройством в программном режиме, необходимо предварительно подготовить его к работе и перевести в режим управления от ЭВМ.

Адрес АЦПУ равен ОС. На АЦПУ могут быть адресованы такие операторы, как оператор вывода текста программы LIST, оператор вывода указателя каталога диска LIST DC, оператор вывода данных на печать PRINT, а также операции консольного вывода. Для адресации этих операторов на АЦПУ необходимо или указать адрес непосредственно в операторе, например

```
LIST/0C
PRINT/0C,  "БЕЙСИК"
```

или перед использованием этих операторов выполнить оператор выбора устройства

```
SELECT  LIST  0C
SELECT  PRINT 0C
```

В последнем случае в качестве устройства вывода для соответствующих операторов будет выбираться АЦПУ (до тех пор, пока не будет задан другой адрес).

Помимо адреса для устройств печати (экран и АЦПУ) в операторе SELECT может быть задан размер строки. По умолчанию строка будет состоять из 80 символов. Максимальная длина строки, выводимой на дисплей, — 80 символов, а на АЦПУ—156 символов. Размер строки задается в скобках после адреса устройства, например

```
 SELECT  LIST  0C (132)
```

или

```
SELECT  LIST  0C (120)
```

Пример 6.1

Введем в машину следующую программу:

```
:10  INPUT  A
:20  PRINT  A/\2
:__
```

Эта программа принимает с клавиатуры число и печатает его квадрат. Выведем текст программы на экран

```
:LIST
10  INPUT  A
20  PRINT  A/\2
:__
```

Для вывода текста программы на АЦПУ следует набрать или

```
:LIST/0C
:__
```

или

```
:SELECT  LIST  0C
:LIST
:__
```

Чтобы программа печатала принятое значение на АЦПУ, можно в явном виде указать адрес для оператора PRINT в программе

```
10  INPUT A
20  PRINT/0C,  A/\2
```

или с помощью оператора SELECT PRINT назначить АЦПУ устройством печати

```
 5  SELECT  PRINT  0C
10  INPUT  A
20  PRINT  A/\2
```

Глава 7. ЦИКЛИЧЕСКИЕ ВЫЧИСЛЕНИЯ

7.1. Организация циклических вычислений с помощью операторов цикла

При программировании часто возникает необходимость многократно повторить одну и ту же последовательность действий. Для организации таких циклов можно воспользоваться уже изученным оператором условного перехода. Рассмотрим для иллюстрации пример программы накопления суммы пяти чисел, которые последовательно вводятся с клавиатуры в ходе выполнения программы.

П р и м е р 7.1

Введем в память ЭВМ следующую программу:

```
:10  REM ЦИКЛ С ОПЕРАТОРОМ УСЛОВНОГО ПЕРЕХОДА
:20  I, S=0
:30  I=I+1
:40  PRINT I ;
:50  INPUT A
:60  S=S+A
:70  IF I<5 THEN 30
:80  PRINT S
```

В строке 20 переменным I и S присваивается значение 0. В строке 30 к значению I прибавляется 1 и в строке 40 выдается на экран результат этого сложения, т. е. 1. Далее в строке 50 с клавиатуры принимается число, которое присваивается переменной A. В строке 60 это значение прибавляется к значению переменной S, т. е. к нулю. В строке 70 значение переменной I сравнивается с числом 5, поскольку значение I равно 1, т. е. меньше 5, то условие в операторе условного перехода удовлетворяется и осуществляется переход к выполнению строки 30.

К значению I снова прибавляется 1, и оно становится равным 2. При выполнении строки 40 это значение печатается. При выполнении строки 50 принимается следующее число, которое присваивается переменной A. При выполнении строки 60 это значение прибавляется к значению переменной S, т. е. к уже накопленной в S сумме 0 и первого числа. В строке 70 осуществляется проверка условия I<5, оно удовлетворяется поскольку I равно 2, и снова выполняется переход к выполнению строки 30 и т. д.

В результате при работе программы пять раз выполняются операторы строк 30—70. Значение переменной I всякий раз увеличивается на 1, т. е. она играет роль счетчика циклов. Каждый раз при печати номера цикла с клавиатуры принимается очередное числовое значение и прибавляется к значению переменной S. При пятом прохождении цикла условие в строке 70 не будет соблюдено и выполнится следующий за оператором условного перехода оператор выдачи значения переменной S на экран.

Выполним эту программу:

```
: RUN
 1?5
 2?4
 3?8
```

```
4?10
5?3
30
:__
```

В Бейсике, как и во многих других языках программирования, имеются специальные операторы для организации циклов — операторы цикла FOR-TO и NEXT. Эти операторы облегчают процесс программирования циклических вычислений. При этом перед последовательностью операторов, которые должны выполняться в цикле, с помощью оператора FOR-TO указывается:
переменная, которая будет служить счетчиком цикла;
начальное значение переменной;
конечное значение переменной.
Рассмотрим пример:

```
:10 FOR  I = 1  TO 4
:__
```

В качестве переменной цикла указана переменная 1, начальное ее значение—1, конечное — 5. За оператором FOR-TO должны следовать операторы, которые будут выполняться в цикле, например

```
:10 FOR  I=1  TO 4
:20 PRINT I,
:30  PRINT 1/\2
```

В данном случае —это операторы печати значения переменной— счетчика циклов и ее квадрата. Запятая в конце оператора печати в строке 20 означает, что следующее значение, выводимое оператором PRINT, должно печататься в той же строке, начиная со следующей 16-символьной зоны экрана. Последовательность операторов должна завершаться оператором NEXT с указанием переменной цикла

```
:10 FOR  I=1  TO 4
:20 PRINT I,
:30 PRINT I^2
:40 NEXT I
```

При выполнении оператора NEXT всякий раз проверяется текущее значение переменной цикла. Если оно меньше конечного значения, заданного в операторе FOR-TO, то к значению переменной добавляется 1 и осуществляется переход к выполнению оператора, следующего за оператором FOR-TO, иными словами, происходит повторное выполнение операторов, находящихся «внутри» операторов цикла. Таким образом, при выполнении указанной программы 5 раз будут выполнены операторы строк 20 и 30:

```
:RUN
   1          1
   2          4
   3          9
   4         16
   :__
```

Циклические вычисления заканчиваются, когда при выполнении оператора NEXT проверка значения переменной цикла покажет, что оно достигло или превзошло заданное конечное значение. В этом случае приращение значения переменной цикла и переход к повторному выполнению операторов не происходят, а выполняется оператор, следующий за оператором цикла.

Пример 7. 2.

В примере 7.1 рассматривалась программа ввода и суммирования 5 чисел, написанная с использованием оператора условного перехода. Ниже приведена аналогичная программа с использованием операторов цикла.

```
:10  REM ЦИКЛ С ОПЕРАТОРАМИ ЦИКЛА
:20  S=0
:30  FOR  I=1  TO  5
:40  PRINT  I;
:50  INPUT  A
:60  S=S+A
:70  NEXT  I
:80  PRINT  S
```

7.2. Общая форма оператора цикла

В общем случае оператор FOR-TO имеет следующую форму:

FOR<числовая переменная> = <а.в.>TO<а.в.> [STEP<а.в.>]

После слова STEP можно задавать произвольную величину приращения счетчика цикла (по умолчанию она принимается равной 1). Эта величина может быть и отрицательной, т. е. можно организовать циклы с убыванием значения переменной цикла. Ниже приведен пример программы, печатающей значения квадратов нечетных чисел от 1 до 9. Для этого начальное значение переменной цикла задается равным 1, конечное — 9, а приращение — 2. В квадрат возводится само значение переменной цикла

```
:10 FOR K=l TO 9 STEP 2
:20 PRINT K/\2
:30 NEXT K
:RUN
1
9
25
49
81
:__
```

Задав начальное значение переменной цикла равным 9, конечное равным 1, а шаг равным — 2, напечатаем те же квадраты в убывающем порядке

```
:10  FOR K=9  TO  1  STEP  -2
:20  PRINT  K/\2
:30 NEXT  K
:RUN
81
49
25
9
1
:__
```

Начальное, конечное значение и приращение переменной цикла допускается задавать в виде арифметических выражений, например

```
FOR  I = A  TO  A+5
FOR Z = X*2 TO Y/\2 STEP Y/10
```

Используя эту возможность, можно модифицировать программу сложения пяти чисел в программу сложения произвольного количества чисел. Для этого в операторе цикла FOR-TO в качестве конечного значения переменной цикла задается значение некоторой переменной, значение которой вводится пользователем при выполнении программы.

П р и м е р 7.3

```
:10    REM СУММИРОВАНИЕ ПРОИЗВОЛЬНОГО КОЛИЧЕСТВА ЧИСЕЛ
:20    INPUT N
:30    S=0
:40    FOR I=1 TO N
:50    PRINT I;
:60    INPUT A
:70    S=S+A
:80    NEXT I
```

При выполнении программы пользователь должен ввести число, означающее количество чисел, которое он хочет сложить (при выполнении оператора ввода в строке 30). Введенное значение присвоится переменной N и таким образом при выполнении оператора FOR-TO (в строке 40) в качестве конечного значения будет фигурировать значение, введенное непосредственно в ходе выполнения программы. Например

```
:RUN
КОЛИЧЕСТВО СЛАГАЕМЫХ? 4
```

```
1?  8
2?  12
3?  4
4?   10
34
:__
```

7.3. Особенности применения операторов цикла

С помощью операторов цикла могут быть организованы сложные последовательности циклических вычислений, когда внутри одного цикла выполняется другой цикл и т. д. Например

```
:10  FOR I=1 TO 4
:20  PRINT I
:30  FOR J=2 TO 4
:40  PRINT I^J,
:50  NEXT J
:60  PRINT
:70  NEXT I
```

Эта программа печатает числа от 1 до 4, значения их квадрата, куба и четвертой степени. Для задания чисел организован цикл с переменной I, значение которой изменяется от 1 до 4.

Внутри этого цикла организован еще один цикл с переменной J, значение которой изменяется от 2 до 4.

Оператор PRINT без параметров в строке 60 используется для перевода строки по окончании печати во внутреннем цикле, поскольку оператор печати во внутреннем цикле заканчивается запятой. В результате выполнения программы на экране появится

```
:RUN
1
1          1        1
2
4          8        16
3
9          27       81
4
16         64       256
:__
```

При вложении циклов друг в друга важно не перепутать последовательность операторов NEXT, недопустим переход по оператору NEXT к повторному выполнению ранее начавшегося цикла до тех пор, пока не будет завершен цикл, начатый позднее. Неправильна, например, следующая программа:

```
:10  FOR  I=1  TO  4
:20  PRINT  I
:30  FOR  J=2  TO  4
:40  PRINT  I^J,
:50  NEXT  I
:60  PRINT
:70  NEXT  J
```

поскольку оператор NEXT I, относящийся к внешнему циклу, стоит ранее оператора NEXT J, относящегося к внутреннему циклу.

При использовании операторов перехода следует помнить, что переход внутрь цикла из произвольного места программы не допускается, так как, не выполнив соответствующий оператор FOR-TO, машина не сможет выполнить оператор NEXT. Недопустимо, в частности

```
:10   GOTO  30
:20   FOR  I=1   TO   10
:30   PRINT  I^3
:40   NEXT  I
```

При обращении к циклу переход должен быть осуществлен к оператору начала цикла. С другой стороны, нежелательно прекращать выполнение цикла, выходя из него без нормального завершения, т. е. без получения переменной цикла значения, равного или большего конечному, и выполнения соответствующего оператора NEXT.

Пример 7.4

В примере 7.3 рассматривалась программа суммирования произвольного количества чисел, которое должно быть подсчитано и введено заранее, до начала ввода самих чисел. Это не всегда удобно. Пусть заранее известно только, что количество суммируемых чисел не может быть больше 100. Напишем программу, которая суммировала бы вводимые числа до тех пор, пока не будет набрано число 0 как признак окончания ввода.

Программа может выглядеть, например, так

```
:10  REM ЦИКЛ СУММИРОВАНИЯ С ОКОНЧАНИЕМ ПО ЧИСЛУ НУЛЬ
:20  S=0
:30  FOR I=1  TO  100
:40  PRINT  I;
:50  INPUT  A
:60  IF  A=0  THEN  90
:70  S=S+A
:80  NEXT  I
:90  PRINT  S
```

В строке 60 осуществляется проверка введенного числа. Если оно равно 0, суммирование прекращается и осуществляется переход к печати суммы.

Однако некорректным является выход из цикла без его завершения. Многократный выход из циклов без их нормального завершения может привести к ошибке при выполнении программы (см. подробнее в гл. 10). Корректной является следующая программа:

```
:10  REM ЦИКЛ СУММИРОВАНИЯ С ОКОНЧАНИЕМ ПО ЧИСЛУ НУЛЬ
:20  S=0
:30  FOR  =1  TO  5
:40  PRINT  I;
:50  INPUT A
:60  IF  A<>  THEN  90
:70  I=100
:80  GOTO  100
:90  S=S+A
:l00  NEXT I
:100  PRINT S
```

В этой программе при вводе 0 переменной цикла искусственно присваивается конечное значение и осуществляется переход к оператору NEXT I, по которому цикл будет нормально завершен.

Глава 8. ИСПОЛЬЗОВАНИЕ МАССИВОВ

8.1. Одномерные и двумерные массивы

В программах часто возникает необходимость выполнять одинаковые операции одновременно со всеми или с отдельными элементами некоторого набора данных. Использование для обозначения этих элементов изученных простых переменных делает программы слишком громоздкими.

Пример 8.1. Пусть на шести складах некоторого предприятия имеются следующие запасы металла:

Номер склада	Количество металла, т
1	3 200
2	650
3	420
4	1 800
5	135
6	300

Для того чтобы учесть поступление 100 тонн металла на склад № 4, необходимо в 4-й строке списка исправить величину запаса (1800 на 1800+100=1900), а также увеличить на 100 суммарный запас (6505 на 6605). Пусть требуется написать программу для решения задачи учета поступлений (выдачи) металла с любого склада из шести имеющихся. Для хранения величины запасов введем семь переменных К1, К2,

…, К6, К7. Переменные К1—К6 соответствуют запасам на складах № 1—6, К7 — суммарным запасам.

```
100  REM ПОДПРОГРАММА УЧЕТА ПОСТУПЛЕНИЙ  (ВЫДАЧ)
110  REM      МЕТАЛЛА  ПО  6  СКЛАДАМ
120  REM С  ИСПОЛЬЗОВАНИЕМ ПРОСТЫХ  ПЕРЕМЕННЫХ
130  REM  ВВОД НОМЕРА  СКЛАДА
140  INPUT  "ВВЕДИТЕ НОМЕР  СКЛАДА  (1-6)",  A
150  REM ПРОВЕРКА  ПРАВИЛЬНОСТИ  ВВОДА
160  IF  ABS(2*A-7)>5  THEN  200
180  IF  A=INT(A)  THEN  240
190  REM СЛУЧАЙ  НЕПРАВИЛЬНОГО ВВОДА
200  PRINT
210  PRINT  "НЕПРАВИЛЬНЫЙ ВВОД"
220  GOTO 140
230  REM ВВОД ПОСТУПЛЕНИЯ (ВЫДАЧИ)
240  INPUT  "СКОЛЬКО МЕТАЛЛА ПОСТУПИЛО  (+ ИЛИ -)",B
250  REM  ВЕТВЛЕНИЕ ПО НОМЕРУ СКЛАДА
260  ON  A  GOTO  280,  320,  360,  400,  440,  480
270  REM  ПОПРАВКА  К1
280  PRINT  "СТАРАЯ ВЕЛИЧИНА ЗАПАСОВ=";К1;"НОВАЯ=";К1+В
290  К1=К1+В
300  GOTO 500
310  REM ПОПРАВКА К2
320  PRINT "СТАРАЯ ВЕЛИЧИНА ЗАПАСОВ"";К2;"Н0ВАЯ=";К2+В
330  К2=К2+В
340  GOTO  500
350  REM  ПОПРАВКА К3
360  PRINT  "СТАРАЯ ВЕЛИЧИНА ЗАПАСОВ=";К3;"НОВАЯ=";К3+В
370  К3=К3+В
380  GOTO 500
390  REM  ПОПРАВКА К4
400  PRINT  "СТАРАЯ ВЕЛИЧИНА ЗАПАСОВ=";К4;"НОВАЯ=";К4+В
410  К4=К4+В
420  GOTO 500
430  REM  ПОПРАВКА К5
440  PRINT  "СТАРАЯ ВЕЛИЧИНА ЗАПАСОВ=";К5;"НОВАЯ=";К5+В
450  К5=К5+В
460  GOTO 500
470  REM  ПОПРАВКА К6
480  PRINT  "СТАРАЯ ВЕЛИЧИНА ЗАПАСОВ = " ;К6 ; "НОВАЯ = " ; К6+В
490  К6=К6+В
500  REM  ПОПРАВКА СУММАРНОГО ЗАПАСА
510  PRINT  "ИТОГО СТАЛО = "; К7+В
520  К7=К7+В
530  GOTO 110
```

В этой программе в строках с номерами от 270 до 490 шесть раз повторяется одна и та же процедура учета поправки, отличаясь всякий раз лишь именем переменной (К1—К6).

Для удобства записи операций с наборами однородных данных в языках программирования используется понятие массива переменных. В Бейсике под массивом в отличие от простой переменной понимается набор переменных, которые имеют общее обозначение и называются элементами массива.

Прежде чем использовать массивы в программе, необходимо объявить, сколько переменных будет в каждом массиве. Это нужно для того, чтобы система отвела необходимую оперативную память для хранения значений элементов массива. Для объявления массива используется уже упоминавшийся оператор DIM, в котором указываются имя каждого массива и в скобках количество элементов массива. Например

```
DIM  A(120),  К(18)
```

В качестве имен массивов используются те же имена, что и для простых переменных, т. е.

A,A0,A1,…, A9,B,B0,B1,..., B9,…, Z9

После имени, как правило, следуют круглые скобки (исключением являются лишь матричные операторы).

Таким образом, простая переменная А2 и массив А2 могут употребляться в одной программе и

обозначать различные величины.

После объявления массива в операторе DIM его имя может встречаться в программе в двух формах.

Во-первых, в форме обозначения массива — имени массива и двух скобок (открывающей и закрывающей), например

A( )

K( )

При использовании в операторе обозначения символьного массива подразумевается, что соответствующая операция выполняется сразу со всеми элементами массива.

Во-вторых, имя массива может встретиться в задании отдельного элемента массива. В этом случае после имени массива в круглых скобках указывается порядковый номер элемента массива, и соответствующая операция будет выполняться только с данным элементом массива. Например

A (5)

K(18)

означают 5-й элемент массива A() и 18-й элемент массива K().

Порядковые номера элементов массива могут задаваться также в виде арифметических выражений, например

A(X+3)

K(4*B—1)

Если арифметические выражения состоят из целой и дробной частей, используется только их целая часть.

Так же как и простые переменные, массивы могут быть целыми. В этом случае к их имени добавляется символ %, например

P%( )

Число элементов в массиве практически ограничивается лишь Объемом оперативной памяти для хранения значений его элементов, но оно не должно превышать 9999.

П р и м е р 8.2. Преобразуем программу из примера 8.1, используя массивы

```
110  REM ПРИМЕР ИСПОЛЬЗОВАНИЯ МАССИВОВ
112  REM ОБЪЯВЛЕНИЕ МАССИВА
115  DIM K(6)
130  REM ВВОД НОМЕРА СКЛАДА
140  INPUT "ВВЕДИТЕ НОМЕР СКЛАДА (1-6)", A
150  REM ПРОВЕРКА ВВЕДЕННОГО НОМЕРА
160  IF ABS(2*A-7)>5 THEN 200
180  IF A=INT(A) THEN 240
190  REM СООБЩЕНИЕ О НЕВЕРНОМ ВВОДЕ
200  PRINT
210  PRINT "НЕВЕРНЫЙ ВВОД"
220  GOTO 140
230  REM ВВОД ВЕЛИЧИНЫ ПОСТУПЛЕНИЯ (ВЫДАЧИ)
240  INPUT "СКОЛЬКО МЕТАЛЛА ПОСТУПИЛО ( + ИЛИ - )",B
250  REM УЧЕТ ПОПРАВКИ
260  PRINT "СТАРАЯ ВЕЛИЧИНА ЗАПАСОВ=", 01(A),
"НОВАЯ=";K(A)+B
270  K(A)=K(A)+B
280  REM ВВЕСТИ СЛЕДУЮЩУЮ ПОПРАВКУ
290  GOTO 110
```

В этом примере в строке 115 объявляется массив с помощью оператора DIM. В операторе DIM K(6) означает, что машина должна создать в оперативной памяти массив из 6 числовых переменных и назвать этот массив именем K().В строке 140 оператор вводит номер склада в переменную A. Результат ввода проверяется на допустимые значения в строках 160—180. В строке 240 объем поступления или выдачи вводится в переменную B.

В строке 260 печатается исходное состояние запасов—K (A). Выражение, заключенное в скобки (A), указывает номер переменной в массиве K(). После того как в строке 140 переменная A получает конкретное значение, K(A) является переменной, где хранится старая величина запасов на выбранном складе. Новая величина запасов равна старой плюс поступление (со знаком) B, т. е. K(A)=K(A)+B. Оператор в строке 270 помещает повое значение на место старого.

Рассмотренные массивы обычно называются одномерными массивами или векторами. Их можно наглядно представить в виде некоторого списка, для задания элемента которого необходимо указать его порядковый номер в списке. Однако это не всегда Удобно. Дело в том, что в ряде приложений обрабатываемые данные естественным образом представляются в виде таблиц, а для задания конкретного элемента указывается номер строки и столбца, на пересечении которых находится искомый элемент.

В связи с этим в Бейсике допускаются так называемые двумерные массивы, или матрицы, элементы которых задаются парой чисел — номером строки и столбца. Двумерный характер массива задается при его объявлении с помощью оператора DIM. В скобках после имени массива в случае двумерного массива указываются два числа через запятую — число строк и столбцов массива. Например, запись

```
DIM P(10,5), E(8,3)
```

означает, что массивы P() и E() объявлены как двумерные. Массив P() состоит из 50 элементов, расположенных в виде таблицы (матрицы) из 10 строк и 5 столбцов, а E() — из 24 элементов, 8 строк и 3 столбцов.

В операциях со всеми элементами двумерного массива используется обычное обозначение массива, например

```
P( )
E( )
```

Если операция должна выполняться с отдельным элементом, указываются номера строки и столбца, например

```
P(3,4)
E(X*2,2)
```

Заметим, что в одной программе имена одномерных и двумерных массивов не должны совпадать.

Использование массивов позволяет выбирать его элементы посредством задания порядкового номера в виде выражения. Всякий раз, когда переменные участвуют в одних и тех же операциях, следует обдумать возможность использования массивов таким . образом, чтобы операцию обработки можно было записать один раз, задав номер обрабатываемого элемента в виде выражения.

8.2. Символьные массивы

Наряду с массивами числовых переменных язык Бейсик «Искры 226» допускает использование массивов символьных переменных. Эти массивы обозначаются точно так же, как и числовые, только после имени массива добавляется символ денежной единицы «⊙». Например, оператор DIM A⊙ (12) дает указание машине создать массив с именем A⊙(), содержащий 12 символьных переменных. Отдельные переменные массива обозначаются как A⊙ (1) ..., A⊙ (12). Аналогично оператор DIM C⊙ (30) создает массив из 30 символьных переменных с именем C⊙ (). Допускается использование одинаковых имен для обозначения числовых и символьных массивов. Например, оператор DIM C5⊙ (30), C5(12) создает два разных массива, первый — символьный, второй — числовой. То же самое касается и простых переменных: C5⊙ и C5 могут использоваться в программе, использующей массив C5⊙ () и C5().

При описании символьных переменных с помощью оператора DIM нужно различать число переменных в списке и длину символьной переменной. Например, DIM A⊙4 означает, что простая символьная переменная A⊙ имеет длину 4 символа.

В то же время оператор DIM A⊙ (4) объявляет символьный список из 4 переменных. Длина каждой переменной этого массива равна 16 символам. Эта длина устанавливается машиной автоматически в тех случаях, когда длина не задается явно в операторе DIM. Результат выполнения оператора DIM A⊙ (4)— создание списка из 4 символьных переменных длиною по 16 символов каждая.

Можно создавать символьные массивы с длиной элемента от 1 до 253 символов, например

```
DIM B⊙ (10)2
DIM C⊙ (14)180
```

Максимальное число переменных в символьном массиве так же, как **и** в случае числовых массивов, равно 7999.

8.3. Массивы и циклы

Массивы переменных позволяют эффективно использовать циклы FOR-NEXT для обработки элементов массивов.

П р и м е р 8.3. Составим программу для ввода величин запасов для шести складов, а также названий пунктов их расположения, используя следующую форму:

№ п/п	Местонахождение	Величина запасов
1 . . . 6		

Будем хранить наименования в символьном массиве А⊙ (), запасы — в числовом массиве К(). Программа примет вид:

```
10  REM  ОБЪЯВЛЕНИЕ МАССИВОВ
20  DIM  K(6),  A⊙ (6)10
30  FOR  P = 1  TO 6
40  PRINT  "СКЛАД №";  P
50  INPUT "НАИМЕНОВАНИЕ",  A⊙ (P)
60  PRINT   ,
70  INPUT "ЗАПАСЫ", К(P)
80  PRINT
90  NEXT P
100  PRINT "ДАННЫЕ ВВЕДЕНЫ"
```

В строке 20 объявляются два массива: числовой К() и символьный А⊙ (), в каждом массиве содержится по 6 переменных. Оператор FOR-TO устанавливает счетчик цикла Р, величина которого меняется от 1 до 6. На каждом шаге цикла (строки 40—70) вводится пара значений — А⊙ (Р) и К(Р).

Пример 8.4. Составим программу, обеспечивающую ввод наименований и величин запасов (как в предыдущем примере), а также учет поправки (поступлений-выдач) для указанного склада.

```
10  REM  ОБЪЯВЛЕНИЕ МАССИВОВ
20  DIM  K(6), A⊙ (6)10, A⊙10
30  FOR  P = 1  TO  6
40  PRINT  "СКЛАД №"";  P
50  INPUT  "НАИМЕНОВАНИЕ",  A⊙ (P)
60  PRINT   ,
70  INPUT  "ЗАПАСЫ",  К(P)
80  PRINT
90  NEXT  P
110  PRINT  "ДАННЫЕ ВВЕДЕНЫ"
130  REM  ОПРЕДЕЛЕНИЕ НАИМЕНОВАНИЯ СКЛАДА
140  INPUT  "ВВЕДИТЕ НАИМЕНОВАНИЕ СКЛАДА",  A⊙
150  REM  ОПРЕДЕЛЕНИЕ НОМЕРА СКЛАДА
160  FOR  P=l  TO  6
170  IF  A⊙ (P)=A  THEN 230
180  NEXT P
190  REM  УКАЗАННОГО СКЛАДА НЕТ В МАССИВЕ
200  PRINT
210  PRINT  "СКЛАДА С УКАЗАННЫМ
МЕСТОРАСПОЛОЖЕНИЕМ В МАССИВЕ НАИМЕНОВАНИЙ НЕТ "
220  GOTO  140
230  REM  СКЛАД НАЙДЕН
240  К=Р:  REM  ЗАПОМИНАНИЕ НОМЕРА
250  P=6
260  NEXT  P
280  REM  ПЕЧАТЬ ТЕКУЩЕГО СОСТОЯНИЯ ЗАПАСОВ
290  PRINT  "СКЛАД";  A⊙ (К), "ЗАПАСЫ=";  К(К)
300  PRINT
310  INPUT  "ВВЕДИТЕ ОБЪЕМ ПОСТУПЛЕНИЙ",  В
```

```
320  K(K) = K(K)+B
330  PRINT  "СКЛАД";  A⊙ (K),"УТОЧНЕННЫЕ ЗАПАСЫ":K(K)
340  PRINT
350  PRINT  "ПОСТУПЛЕНИЕ УЧТЕНО"
360  REM  НАЧАЛЬНАЯ УСТАНОВКА ПЕРЕМЕННЫХ И.
ВОЗВРАТ ДЛЯ НОВОГО ВВОДА
370  A⊙ = "  "
380  B=0
390  GOTO 140
```

В отличие от предыдущих примеров здесь пользователь задает нужный склад введением с клавиатуры не номера, а наименования места нахождения склада. Затем в программе с целью нахождения номера просматривается массив наименований A⊙ (). Этот поиск осуществляется в строках 160—180. Счетчик цикла P используется, как обычно, для указания переменной массива. Как только выполняется сравнение в строке 170, происходит выход из цикла на строку 230. Здесь запоминается значение P, при котором произошел выход из цикла, и цикл искусственно завершается (строки 240—260). Таким образом, переменная K содержит результат поиска — номер заданного склада. Отметим при этом, что нормальное завершение цикла FOR-NEXT происходит без нахождения заданного наименованием A⊙ склада, после чего печатается сообщение «не найдено» (строка 210) и происходит переход на повторный ввод A⊙. В строках 280—350 происходит прием с клавиатуры величины поступлений и расчет новой величины запасов. Значения переменных A⊙ и B стираются в строках 370 и 380, так как иначе случайное нажатие клавиши CR/LF может привести к неверным результатам из-за сохранения предыдущих значений при повторении процесса вычислений.

Циклы FOR-NEXT и массивы переменных широко используются в технике программирования различных операций. Приведем несколько примеров.

Пример 8.5. Присвоить переменным числовой массив постоянного значения 100 можно, составив следующую программу:

```
10  DIM  A(30)
20  FOR  P=l  TO  6
30  A(P)=100
40  NEXT  P
```

Пример 8.6. Ниже представлена программа для расчета суммарных запасов по шести складам. Следует напечатать таблицу вместе с итогом.

```
5   DIM  A⊙(6),  K(6)
10  REM  ВВОД ИСХОДНЫХ ДАННЫХ
20  REM  И  РАСЧЕТ СУММЫ ЗАПАСОВ
30  S = 0
40  FOR  P = l  TO  6
50  INPUT  "НАИМЕНОВАНИЕ". A⊙ (P)
60  INPUT  "ЗАПАСЫ",  K(P)
70  S=S+K(P)
80  NEXT  P
90  REM  ПЕЧАТЬ ТАБЛИЦЫ
100  PRINT:  PRINT
110  PRINT "НОМЕР ПО ПОРЯДКУ",  "НАИМЕНОВАНИЕ",
"ЗАПАСЫ"
120  FOR  P=l  TO  6
130  PRINT  P,  A⊙(P),  K(P)
140  NEXT  P
150  PRINT
160  PRINT  "ВСЕГО"  ,  , S
```

Пример 8.7. С помощью программы, представленной ниже, ввести значения массива A(), состоящего из десяти переменных, Затем следует напечатать наименьшую переменную массива.

```
10  DIM  A(10)
20  FOR  P=l  TO  10
30  PRINT  P,
40  INPUT  A(P)
```

```
50  NEXT  P
60  REM  ПОИСК НАИМЕНЬШЕГО ЧИСЛА
70  REM  M-ПЕРЕМЕННАЯ ДЛЯ ХРАНЕНИЯ НАИМЕНЬШЕГО
ЧИСЛА
80  M=A(1)
90  FOR  P=2  TO  10
100  IF  A(P)>=M  THEN  120
110  M=A(P)
120  NEXT  P
130  REM  ПЕЧАТЬ НАИМЕНЬШЕГО ЧИСЛА ИЗ СПИСКА
140  PRINT  "МИНИМАЛЬНОЕ ЧИСЛ0=";  M
```

В этом примере переменной M присваивается начальное значение— значение первой переменной массива A(). Затем просматривается в цикле оставшаяся часть массива — переменные со второй по десятую. Если среди них встретится переменная, меньшая чем M, то ее значение записывается в переменную M.

Если бы нужно было дополнительно напечатать кроме наименьшего значения еще и номер соответствующей этому значению переменной, то предыдущее задание следовало бы сформулировать так: выбрать переменную для хранения номера наименьшей переменной, например E; строку 80 заменить на строку

```
80  M=1:  E=1
```

а строку 110 на строку

```
110  M = A(P):  E=P
```

и, наконец, строку 140 на строку

```
140  PRINT  "МИНИМАЛЬНОЕ ЧИСЛО=";  M;  "НОМЕР=",  E
```

В заключение приведем пример сортировки элементов числового массива A() в возрастающем порядке.

Пример 8.8

```
10   REM  ПРОГРАММА СОРТИРОВКИ ЧИСЛОВОГО МАССИВА A( )
20   REM  В ВОЗРАСТАЮЩЕМ ПОРЯДКЕ
30   DIM  A(100)
40   PRINT  "ВВОД ФАКТИЧЕСКОГО ЧИСЛА ПЕРЕМЕННЫХ";
50   NPUT  C
60   REM  ВВОД ИСХОДНЫХ ДАННЫХ
70   FOR  P = 1  TO  C
80   INPUT  A(P)
90   NEXT  P
100  REM  ПЕЧАТЬ ИСХОДНОГО МАССИВА A( )
110  PRINT:  PRINT
120  FOR  P = 1  TO  C
130  PRINT  A(P)
140  NEXT  P
150  REM  СОРТИРОВКА
160  FOR  P=l  TO  C-1
170  FOR  P1=2  TO  C .
180  IF  A(P)<=A(P1)  THEN  230
190  REM  ПОМЕНЯТЬ ВЕЛИЧИНЫ  A(P)  и  A(P1)
200  M=A(P1)
210  A(P1)=A(P)
220  A(P)=M
230  NEXT  P1
240  NEXT  P
250  ПЕЧАТЬ ОТСОРТИРОВАННОГО МАССИВА A( )
260  FOR  P=1  TO  C
270  PRINT  A(P)
280  NEXT  P
```

В строках 70—90 в цикле вводятся значения массива A(). В строках 110—140 печатается исходный массив, чтобы его можно было сравнить с отсортированным. Сама процедура сортировки состоит из двух вложенных циклов, при этом вся обработка реализуется во внутреннем цикле. Пусть, например, неотсортированный массив A() состоит из десяти элементов (C равно 10). В начале работы внешнего

цикла P равно 1, а A(P) равно первому числу из массива A(). Во время работы внутреннего цикла начинают просматриваться остальные числа массива, кроме первого, т. е. со 2-го по 10-е. Если окажется, что текущее просматриваемое число больше или равно A(1), то происходит переход к просмотру следующего числа. Если же условие в строке 180 не выполняется, т. е. текущее просматриваемое число меньше A(1), то это число меняется с A(1) местами. Для того чтобы поменять значения двух переменных, приходится использовать вспомогательную переменную M. Процедура обмена реализуется в строках 200—220. Таким образом, просмотрев числа со 2-го по 10-е, получим в A(1) наименьшее число в массиве A(). Этот ход рассуждений аналогичен тому, который использовался при нахождении наименьшего числа в массиве.

После завершения первого выполнения внутреннего цикла счетчик внешнего цикла станет равным 2. A(P) будет равно второму числу из массива A(). Во время работы внутреннего цикла будут просматриваться числа из массива A() с 3-го по 10-е и сравниваться со значением переменной A(2). Если во время просмотра найдутся числа, меньшие чем A(2), то самое меньшее будет записано в A(2) и т. д. По окончании работы внутреннего цикла в A(2) окажется меньшее из оставшихся 9 чисел массива A().

Описанная процедура повторяется C—1, т. е. 9 раз. При выполнении последнего внешнего цикла P станет равным 9. На предыдущем шаге в A(8) было помещено наименьшее из оставшихся 3 чисел. В A(P) будет находиться 9-е число массива. Работа внутреннего цикла заключается в сравнении A(9) и A(10). Если A(9)>A(10), то они поменяются местами. На этом сортировка массива A() заканчивается. Описанный метод называется обменной сортировкой.

Для опробования программы сортировки нужно ввести с клавиатуры исходные данные либо, например, вставить вместо процедуры ввода (строки 70—90) подпрограмму формирования списка случайных чисел.

```
70 FOR P=l TO C
80 A(P)=INT(RND(P)*10000+1)
90 NEXT P
```

Глава 9. ВЫЧИСЛЯЕМЫЕ ПЕРЕХОДЫ

9.1. Оператор вычисляемого перехода

Для выполнения перехода по какому-либо условию в программах используется рассмотренный ранее оператор условного перехода. При этом допускается проверка сразу одного или даже нескольких условий и при их выполнении осуществляется переход к соответствующей строке программы. В ряде случаев, однако, возникают ситуации, при которых нужно проверить несколько условий и в зависимости от этого осуществлять переходы к различным строкам.

Пример 9.1

В следующем фрагменте программы по желанию пользователя выполняются различные действия со значениями переменных A и B.

```
500 REM ВЫБОР ДЕЙСТВИЯ
510 PRINT "1 - A + B"
520 PRINT "2 - A-B"
530 PRINT "3 - A*B"
540 PRINT "4 - A/B"
550 PRINT "5 - A^B"
560 PRINT "6 - ЗАКОНЧИТЬ"
570 PRINT
580 INPUT "ВВЕДИТЕ НОМЕР НЕОБХОДИМОГО
ДЕЙСТВИЯ", C
590 REM ПЕРЕХОД К ЧАСТИ ПРОГРАММЫ, РЕАЛИЗУЮЩЕЙ
ВЫБРАННОЕ ДЕЙСТВИЕ
600 IF C=l THEN 660
610 IF C=2 THEN 680
620 IF C=3 THEN 700
630 IF C=4 THEN 720
640 IF C=5 THEN 740
650 IF C=0 THEN 760
655 PRINT "НЕПРАВИЛЬНЫЙ ВВОД": GOTO 5 40
```

```
660  PRINT "СУММА  -"; A+B
670  GOTO  540
680  PRINT  "РАЗНОСТЬ  -"; A-B
690  GOTO  540
700  PRINT  "ПРОИЗВЕДЕНИЕ  -"; A*B
710  GOTO  540
720  PRINT  "ЧАСТНОЕ  -"; A/B
730  GOTO  540
740  PRINT  "СТЕПЕНЬ  -"; A^B
750  GOTO  540
760  END
```

При выполнении оператора INPUT в строке 580 пользователь должен ввести число — номер требуемого действия. В строках 600—650 осуществляются проверка переменной С и в зависимости от ее значения переход к части программы, реализующей вывод результатов соответствующих действий, или к оператору окончания программы END.

Проверка значения переменной С и переход при удовлетворении проверяемого условия осуществляются с помощью шести последовательных операторов IF-THEN. Если не удовлетворяется ни одно из условий, выводится сообщение об ошибке (строка 655) и осуществляется повторный выбор действия. В данном примере приводится часто встречающийся в программах случай, когда в зависимости от значения арифметического выражения (в примере— в зависимости от значения переменной С) необходимо совершить переход к различным частям программы.

Оператор ON-GOTO позволяет задать серию таких переходов в зависимости от значения арифметического выражения. В частности, в приведенном фрагменте программы строки 600—650 могут быть заменены на одну строку с оператором ON-GOTO

```
600 ON   C   GOTO  660,680,700,720,740,760
```

При выполнении этого оператора, если значение С равно 1, осуществляется переход к выполнению первой из перечисленных строк, т. е. строки 660. Если значение С равно 2, то осуществляется переход к строке 680 и т. д. Если целая часть значения С меньше единицы или больше 6, то выполняется оператор, следующий за оператором ON-GOTO, т. е. оператор в строке 755.

В общем случае в операторе ON-GOTO после слова ON задается арифметическое выражение, а после GOTO перечисляются через запятую номера строк, например

```
ON A*5+3  GOTO  1250,185,80
```

При выполнении оператора сначала вычисляется целая часть значения арифметического выражения, а затем осуществляется переход к выполнению строки с соответствующим порядковым номером в списке, равным вычисленной целой части. Иными словами, если целая часть равна *i,* то осуществляется переход к выполнению *i*-й строки из списка. Если значение целой части выражения меньше 1 или больше количества номеров в списке, выполняется следующий за ON-GOTO оператор.

9.2. Особенности применения оператора вычисляемого перехода

В примере 9.1 при замене операторов условного перехода IF-THEN на оператор вычисляемого перехода ON-GOTO было, казалось бы, осуществлено эквивалентное преобразование программы. Однако имеется одно существенное различие в функционировании этих двух вариантов программ. Если значение переменной С дробное, скажем 2.6, то в варианте с операторами IF-THEN будет выдано сообщение об ошибке, а в варианте с оператором ON-GOTO выполнено вычитание.

Поскольку в операторе ON-GOTO для определения номера строки берется целая часть арифметического выражения, то все числа, целая часть которых одинакова, зададут переход к одному и тому же номеру строки. В данном примере значение $1 \le C < 2$ задает переход к строке 660, значение $2 \le C < 2$ — к строке 680 и т.д.

Если такое различие существенно, то следует предварительно проверять, целое ли число. В примере 9.1 для этого достаточно добавить строку

```
595   IF   INT (C)  <>C   THEN   655
```

Иногда, напротив, можно использовать эту особенность оператора ON-GOTO для задания условия на нахождение числового значения в определенном диапазоне. Например, требуется совершить переход к программной строке 110, если значение переменной А находится в диапазоне

1≤A<4

Это можно сделать таким образом

```
ON  A  GOTO  110,110,110
```

Переменная, задающая переход к выполнению различных частей программы, может не принимать значений, соответствующих порядковым номерам 1, 2, ... и т. д. В этом случае, чтобы воспользоваться оператором ON-GOTO, следует попытаться сформировать арифметическое выражение, значения которого лежат в нужных пределах.

Если требуется осуществить проверку на нахождение значения переменной в диапазоне, начинающемся не с 0, то достаточно прибавить или вычесть соответствующую константу. Например, следует осуществить переход к строке 110, если значение переменной А лежит в диапазоне

100≤A<103

тогда можно записать

```
ON  A—99  GOTO  110,110,110
```

Если переход к различным частям программы должен осуществляться в зависимости от нахождения значения переменной в одном из диапазонов, заданных с одинаковым шагом, то в качестве выражения в операторе ON-GOTO можно использовать частное от деления переменной на величину диапазона с добавлением или вычитанием соответствующей константы.

Пример 9.2

Пусть в программе требуется совершать переход к различным строкам, если значение переменной А находится в следующих диапазонах:

0≤A<800 — переход к строке 900;

800≤A < 1600 —переход к строке 800;

1600≤A<2400 — переход к строке 700;

2400≤A<3000 — переход к строке 600.

Если A≥3000, должен выполняться следующий оператор. В этом случае оператор вычисляемого перехода будет выглядеть так:

```
ON  A/800+1  GOTO  110,110,110
```

Рассмотрим еще один пример формирования арифметического выражения для оператора ON-GOTO.

Пример 9.3

Пусть в программе требуется выполнять три различные операции в зависимости от значения переменной К, которое может быть меньше, больше или равным нулю. Для представления знака в числовой форме удобно использовать стандартную функцию Бейсика SGN. Она принимает значение: — 1 — при отрицательном, 0 — при нулевом и +1 —при положительном аргументе. Выражение SGN(K) + 1, следовательно, будет иметь значения 0, 1 и 2 соответственно и может быть использовано в операторе вычисляемого перехода.

```
100  REM  СЕГМЕНТ, ИСПОЛЬЗУЮЩИЙ ФУНКЦИЮ SGN ДЛЯ
ВЫЧИСЛЯЕМОГО ПЕРЕХОДА
110  ON  SGN(K)  +  1  GOTO  170, 210
120  REM  ОПЕРАЦИЯ  ПРИ  K<0
. . . . .
170  REM  ОПЕРАЦИЯ  ПРИ  K=0
. . . . .
210  REM  ОПЕРАЦИЯ  ПРИ  K>0
```

Глава 10. ОРГАНИЗАЦИЯ ПОДПРОГРАММ

10.1. Операторы перехода в подпрограмму и возврата из подпрограммы

При составлении программ очень часто возникает необходимость многократного выполнения некоторых операций. Конечно, можно всякий раз повторно записывать в программе одни и те же последовательности операторов. Однако это приводит к увеличению текста программы и занимаемой ею памяти, повышает трудоемкость программирования. В связи с этим в Бейсике предусмотрен ряд

специальных средств, позволяющих эффективно программировать многократно повторяющиеся операции.

Одно из таких средств — операторы цикла, описанные в гл. 7. Другое средство — использование массивов при программировании одинаковых операций с большим числом переменных (см. гл. 8). Если в операторе указано имя массива, то соответствующая операция выполняется сразу со всеми его элементами. Если же указан элемент массива с переменными в качестве индексов, то один и тот же оператор может всякий раз выполнять операции с различными элементами массива в зависимости от значений переменных. Еще одним средством программирования повторяющихся операций является использование подпрограмм.

Идея использования подпрограмм заключается в следующем. Если известно, что в различных местах программы понадобится исполнять одну и ту же последовательность операторов, то эта последовательность оформляется как отдельная вспомогательная программа, называемая подпрограммой. В тех местах основной программы, где необходимо выполнение этой последовательности, помещаются лишь обращения к подпрограмме, обеспечивающие ее выполнение, и возврат к продолжению основной программы.

Подпрограммой в Бейсике может являться любая последовательность строк программы, завершающаяся специальным оператором конца подпрограммы RETURN. Обращение к подпрограмме может осуществляться из любого места программы с помощью оператора перехода к выполнению подпрограммы GOSUB

GOSUB <номер строки>

При выполнении этого оператора система запоминает адрес следующего оператора, а затем, так же как при выполнении оператора безусловного перехода GOTO, переходит к выполнению строки с указанным номером. Выполнение строк подпрограммы продолжается до тех пор, пока не встретится оператор RETURN. По этому оператору система возвращается к выполнению программы в соответствии с хранимым адресом, т. е. начиная с оператора, следовавшего за оператором GOSUB, а сам адрес возврата стирается.

Рассмотрим два примера, иллюстрирующие применение подпрограмм в Бейсике.

Пример 10.1. Пусть в программе в различных местах необходимо выводить различную информацию на экран дисплея и при этом каждый раз очищать экран дисплея и печатать один и тот же заголовок. В этом случае операцию очистки дисплея и печати заголовка удобно оформить подпрограммой

```
100  REM  ПРОГРАММА, ИСПОЛЬЗУЮЩАЯ ПОДПРОГРАММУ
110  REM  ПЕЧАТИ ЗАГОЛОВКА
. . .
500  REM  ТРЕБУЕТСЯ ВЫВОД ЗАГОЛОВКА
510 GOSUB 1 000
. . .
990  REM  ПОДПРОГРАММА ВЫВОДА ЗАГОЛОВКА
1000  PRINT  HEX(03)  :  REM  ОЧИСТКА ЭКРАНА
1010  PRINT  "ЗАГОЛОВОК"
1020  RETURN
```

Пример 10.2. Пусть в программе в различных местах требуется вводить числа, осуществляя контроль вводимых значений: числа должны находиться в диапазоне от 1 до 99 и содержать не более двух знаков после десятичной точки. В противном случае ввод повторяется.

```
100  REM  ПРОГРАММА, ИСПОЛЬЗУЮЩАЯ ПОДПРОГРАММУ
110  REM  ВВОДА ЧИСЕЛ С ПРОВЕРКОЙ
. . .
300  REM  ТРЕБУЕТСЯ ВВОД ЧИСЛА
310  GOSUB  900
. . .
350  REM  ТРЕБУЕТСЯ ВВОД ЧИСЛА
360  GOSUB  900
. . .
900  REM  ПОДПРОГРАММА ВВОДА
910  INPUT  "ЧИСЛО  ОТ  1  ДО  99  (НЕ БОЛЕЕ 2-Х ЗНАКОВ
ПОСЛЕ ДЕСЯТИЧНОЙ ТОЧКИ",  A
920  IF  ABS(2*A-100)>98  THEN  910
```

```
930 IF A*100<>INT(A*100) THEN 910
940 REM A - НАХОДИТСЯ В ТРЕБУЕМОМ ДИАПАЗОНЕ
950 RETURN
```

При использовании подпрограмм необходимо следить, чтобы работа каждой подпрограммы заканчивалась выполнением оператора RETURN. Будет зафиксирована ошибка, если по ходу исполнения встретится оператор RETURN, а оператор GOSUB до этого не исполнялся. Подпрограммы могут быть вложены одна в другую. Число вложений ограничивается только размером свободной памяти машины, в которой помещаются адреса возврата. Очистка этой области производится при выполнении операторов RUN или CLEAR.

10.2. Оператор окончания подпрограммы без возврата

Иногда после выполнения подпрограммы нет необходимости возвращаться к оператору, следующему за оператором перехода к подпрограмме.

Пример 10.3. В этом примере используется подпрограмма ввода положительных чисел. При вводе значения, равного нулю, программа заканчивает работу.

```
100 REM ВЫЧИСЛЕНИЕ ДЕСЯТИЧНЫХ ЛОГАРИФМОВ
110 GOSUB 150
120 PRINT "ЛОГАРИФМ ОТ"; B;"=";LOG(B)/LOG(10)
130 GOTO 110
140 REM ПОДПРОГРАММА ВВОДА
150 INPUT "ЧИСЛО"; B
160 IF B<0 THEN 150
170 F B=0 THEN 190
180 RETURN
190 RETURN CLEAR
200 PRINT "ВЫЧИСЛЕНИЯ ОКОНЧЕНЫ"
210 INPUT "ПРОДОЛЖИТЬ - 1"; A
220 F A=l THEN 110
230 END
```

Если при выполнении подпрограммы вводятся отличные от нуля числа, то при вводе положительных чисел происходят возврат из подпрограммы и печать результата вычисления десятичного логарифма от вводимого числа. Если введенное число отрицательное, в подпрограмме повторяется оператор ввода. Если введенное число равняется нулю, то переход в подпрограмме осуществляется к программной строке 190, в которой записан оператор RETURN CLEAR.

При выполнении оператора RETURN CLEAR происходит стирание адреса возврата и собственно выполнение подпрограммы заканчивается. Переход к оператору, следующему за оператором GOSUB, не производится, а выполняется следующий за оператором RETURN CLEAR оператор. В примере 10.3 выполняется оператор в программной строке 200 и т. д. Следует отметить, что, так же как и при выполнении оператора RETURN, после выполнения оператора RETURN CLEAR машина уже «не помнит», что выполнялась подпрограмма.

Подпрограммы могут вкладываться одна в другую, т. е. из одной подпрограммы возможно обращение к другой.

Машина может хранить сразу много адресов возврата. При выполнении оператора RETURN список этих адресов уменьшается на один адрес и происходит возврат к оператору, следовавшему за последним исполнявшимся оператором GOSUB. При выполнении оператора RETURN CLEAR список адресов возврата тоже уменьшается на один адрес, соответствующий последнему обращению к данной подпрограмме. Если оператор RETURN CLEAR выполняется в последней вложенной подпрограмме, то при исполнении очередного оператора RETURN происходит возврат к оператору, следующему за предпоследним исполненным оператором GOSUB.

П р и м е р 10.4. В данном примере программа использует подпрограмму вычисления факториала.

```
100 REM ПРОГРАММА, ИСПОЛЬЗУЮЩАЯ .
110 REM ПОДПРОГРАММУ ВЫЧИСЛЕНИЯ ФАКТОРИАЛА
120 PRINT "ВЫЧИСЛЕНИЕ ФАКТОРИАЛА"
130 INPUT "ВВЕДИТЕ ПОЛОЖИТЕЛЬНОЕ ЦЕЛОЕ ЧИСЛО <= 1", A
140 P = 1
150 GOSUB 240
```

```
160  PRINT  "ФАКТОРИАЛ="; P
170  GOTO  120
190  REM  ПОДПРОГРАММА, ВЫЧИСЛЯЮЩАЯ ФАКТОРИАЛ
200  IF A  =  1  THEN  250
210  P=P*A ;
220  A=A-1
230  RETURN  CLEAR
240  GOSUB  200
250  RETURN  CLEAR
260 RETURN
```

При выполнении указанной программы после ввода числа в строке 130 переменной P присваивается значение единицы и происходит переход к подпрограмме вычисления факториала. В первой исполняемой строке подпрограммы записан оператор перехода к подпрограмме в строке 200. Таким образом, после выполнения оператора GOSUB 200, записанного в строке 240, начинается выполнение вложенной в первую подпрограмму второй подпрограммы. В этой подпрограмме исходное число сравнивается с единицей и при несовпадении осуществляется умножение результата на значение исходного числа, а исходное число уменьшается на единицу. Затем выполняется оператор RETURN CLEAR в программной строке 230, и вторая подпрограмма заканчивается. Оператор GOSUB в программной строке 240 вновь осуществляет выполнение второй подпрограммы. Выполнение подпрограммы продолжается, таким образом, многократно, а в переменной P накапливается результат произведения чисел, убывающих от значения переменной A до единицы. В этом случае при выполнении оператора условного перехода в строке 200 осуществляется переход к программной строке 250. Оператор RETURN CLEAR, записанный в этой строке, заканчивает выполнение второй подпрограммы. Оператор RETURN в следующей строке также заканчивает выполнение второй подпрограммы. Оператор RETURN в стоке 260 заканчивает выполнение первой подпрограммы и осуществляет переход к оператору печати результата в основной программе.

10.3. Оператор вычисляемого перехода к подпрограммам

Оператор вычисляемого перехода к подпрограммам ON-GOSUB аналогичен оператору вычисляемого перехода ON-GOTO. Использование оператора ON-GOSUB позволяет осуществлять переход к одной из подпрограмм, начинающихся с перечисленных в операторе номеров строк. Оператор ON-GOSUB имеет следующую общую форму:

ON<а.в.>GOSUB<номер строки>[,<номер строки>...]

П р и м е ч а н и е . Многоточие означает возможность многократного повторения предшествующего элемента.

При исполнении оператора ON-GOSUB вычисляется значение арифметического выражения, целая часть которого используется при определении порядкового номера строки из списка номеров строк, к которой осуществляется переход. Если значение целой части выражения в операторе ON-GOSUB меньше единицы или больше количества номеров в строке в списке, выполняется оператор, следующий за оператором ON-GOSUB. Переход к подпрограмме осуществляется аналогично переходу по оператору GOSUB. В качестве адреса возврата запоминается адрес следующего за оператором ON-GOSUB оператора программы и выполняется оператор перехода к подпрограмме, начинающейся с вычисленной строки. По окончании работы подпрограммы, т. е. после исполнения оператора возврата RETURN, выполнение программы продолжается с оператора, следующего за оператором ON-GOSUB.

П р и м е р 10.5. В программе, осуществляющей подсчет суммы вводимых чисел и вычисление среднего значения, требуется выводить результат либо на экран дисплея, либо на печатающее устройство. Для выбора необходимого выводного устройства используется аппарат подпрограмм.

```
100  REM  ПРОГРАММА, ИСПОЛЬЗУЮЩАЯ ПОДПРОГРАММЫ
110  REM  ВЫБОРА УСТРОЙСТВА ВЫВОДА
120  REM  ВВОД ЧИСЕЛ
130  INPUT  "КОЛИЧЕСТВО ЧИСЕЛ",  A
140  S=0
150  FOR  K=1  TO  A
160  INPUT  "ОЧЕРЕДНОЕ ЧИСЛО",  X
170  S=S+X
180  NEXT  K
190  INPUT  "ВЫВОД НА ЭКРАН  -1,  ВЫВОД НА ПЕЧАТЬ -2", B
```

```
200  REM  ПЕРЕХОД К ПОДПРОГРАММАМ ВЫБОРА УСТРОЙСТВА
210  ON  B  GOSUB  300,520:  GOTO  190
220  PRINT  "СУММА=";S,  "СРЕДНЕЕ=";  S/A
230  GOTO  130
300  SELECT  PRINT  05
310  RETURN
320  SELECT  PRINT  OC
330  RETURN
```

После ввода и подсчета суммы введенных чисел программа переходит в строке 190 к выполнению оператора ввода, при исполнении которого вводится число, определяющее устройство вывода результатов выполнения программы. По оператору ON-GOSUB происходит переход к подпрограммам выбора устройства. После исполнения подпрограммы результаты выполнения программы выводятся на выбранное устройство вывода и осуществляется переход к началу программы. Если при выполнении оператора ввода в строке 190 будет введено число, значение которого лежит вне диапазона от единицы до двух, переход к подпрограммам не произойдет и выполнится оператор перехода к строке 190.

10.4. Помеченные подпрограммы и передача параметров в подпрограмму

Обращения к подпрограммам, рассмотренные в предыдущих разделах, осуществлялись с помощью оператора перехода к подпрограмме по номеру строки. Используя такое обращение, при составлении программы необходимо знать, с какой строки начинается необходимая подпрограмма. Если в процессе составления программы использовалась операция перенумерации программных строк с помощью оператора RENUMBER, то необходимо после такой операции просматривать программу для нахождения номера строки, с которой начинается подпрограмма. Естественно, это требуется делать в том случае, когда при дальнейшем изменении или дополнении текста программы нужно использовать уже имеющуюся подпрограмму, потому что операция перенумерации изменяет как номера строк программы, так все номера строк, заданные в операторах GOSUB, сохраняя запрограммированную логику переходов. Таким образом, обращение к подпрограммам с использованием операторов перехода к начальной строке подпрограммы не всегда является удобным. Кроме того, такое обращение затрудняет возможность программирования «сверху вниз», при котором в процессе составления программы осуществляется обращение к подпрограммам, еще не написанным или еще не введенным. А программирование «сверху вниз» в отличие от программирования «снизу вверх», когда сначала разрабатываются все используемые подпрограммы, а уже затем составляется основная программа, имеет ряд очевидных преимуществ. Преимущества программирования «сверху вниз» определяются, как правило, более быстрым и эффективным способом составления программ. Действительно, при составлении достаточно сложной программы заранее трудно бывает определить, какие подпрограммы будут необходимы, а также установить порядок их использования и все необходимые функции, выполняемые подпрограммами. При программировании «сверху вниз» достаточно (при необходимости) записывать в тексте основной программы обращения к подпрограммам и отмечать для себя комментариями необходимые действия, выполняемые подпрограммами. После составления текста основной программы можно просмотреть весь набор необходимых подпрограмм и далее приступить к их составлению. В свою очередь в этих подпрограммах может оказаться возможным использование однотипных операций, и таким образом снова можно определить необходимые подпрограммы, к которым организуется обращение из составляемых подпрограмм. Процесс программирования продолжается до тех пор, пока не будут составлены самые «нижние» подпрограммы. Понятно, что такой способ программирования обеспечивает, как правило, достаточно быстрое написание сложных программ с большим количеством вложенных подпрограмм, т. е. получение минимального по объему текста программы. Кроме того, при таком способе программирования на любом этапе составления подпрограмм возможно проводить отладочные расчеты по всей задаче, подставляя результаты выполнения еще не написанных подпрограмм вместо обработки в этих подпрограммах. Проведение такой отладки гарантирует работоспособность всей программы после составления и отладки самых «нижних» подпрограмм.

Средства языка машин предоставляют возможность именования подпрограмм. В качестве имени подпрограммы используются целые числа от 0 до 255. Таким образом, возможно в одной программе использовать до 256 различных подпрограмм или, точнее, различных помеченных входов в подпрограммы. Для того чтобы пометить подпрограмму (или вход в подпрограмму), в начальной строке

подпрограммы записывается оператор DEFFN' с указанием имени программы. Например, оператор

```
1000 DEFFN' 152
```

определяет начало подпрограммы с именем (номером) 152. Для перехода к такой подпрограмме используется оператор GOSUB' с указанием имени соответствующей подпрограммы. Например, переход к подпрограмме 152 записывается как GOSUB' 152. Этот оператор перехода к подпрограмме с номером аналогичен оператору перехода к подпрограмме с номером строки. Различие операторов состоит в том, что при выполнении оператора GOSUB' сначала просматривается текст программы и ведется поиск оператора DEFFN' с тем же номером, что и в операторе перехода к подпрограмме. Затем начинается выполнение подпрограммы, т. е. запоминается адрес возврата и выполняется оператор, следующий за оператором DEFFN', и т. д. Возврат из помеченной подпрограммы осуществляется так же, как и из непомеченных подпрограмм с помощью операторов RETURN и RETURN CLEAR. Нумерацию помеченных подпрограмм можно проводить в любом порядке, так как номер подпрограммы в действительности обозначает только ее имя. Используя операторы DEFFN' и GOSUB', можно располагать подпрограммы и обращения к ним, не привязываясь к определенным номерам строк программ, т. е. можно использовать описанный выше способ программирования «сверху вниз».

Общая форма операторов определяется следующим образом:

```
DEFFN' <целое число> (<список параметров>)
```

где целое число должно принадлежать диапазону 0—255. Оператор DEFFN' должен быть первым оператором в программной строке. Размещение в тексте программы операторов DEFFN' и соответствующих операторов обращения GOSUB' значения не имеет, они могут встречаться в программных строках до или после текста подпрограммы. Если при выполнении программы без исполнения соответствующего оператора GOSUB' встречается оператор DEFFN', он не влияет на ход выполнения программы, так как единственная функция этого оператора — отметить вход в подпрограмму.

Пример 10.6. Пусть в различных местах программы требуется определять сумму элементов числового массива А, используемого в основной программе, накапливать полученные суммы и получать средние значения накопленных сумм.

```
10 DIM A(100)
. . .
100 REM ПЕРВОЕ ОБРАЩЕНИЕ К ПОДПРОГРАММЕ ВЫЧИСЛЕНИЯ И НАКОПЛЕНИЯ СУММ 110 60SUB '100
. . .
200 REM ПОВТОРНЫЕ ОБРАЩЕНИЯ К ПОДПРОГРАММЕ НАКОПЛЕНИЯ СУММ
210 GOSUB '101
. . .
250 GOSUB '101
. . .
500 REM ОБРАЩЕНИЕ К ЧАСТИ ПОДПРОГРАММЫ, ВЫЧИСЛЯЮЩЕЙ СРЕДНЕЕ ЗНАЧЕНИЕ НАКОПЛЕННЫХ
СУММ
510 GOSUB '102
. . .
1000 REM НАЧАЛЬНЫЙ ВХОД ПОДПРОГРАММЫ ВЫЧИСЛЕНИЯ
СУММ
1010 DEFFN '100
1020 REM ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕННЫХ, ИСПОЛЬЗУЕМЫХ
ДЛЯ ХРАНЕНИЯ НАКОПЛЕННЫХ СУММ И КОЛИЧЕСТВА ОБРАЩЕНИЙ
1030 S1, N=0
1040 REM ВХОД ПОВТОРНЫХ ОБРАЩЕНИЙ
1050 DEFFN '101: S = 0
1060 FOR K=l TO 100: S=S+A(K)
1070 N=N+1
1080 S1=S1+S: RETURN
1090 REM ВХОД ПОДПРОГРАММЫ ВЫЧИСЛЕНИЯ СРЕДНЕГО
ОТ НАКОПЛЕННОЙ СУММЫ
1100 DEFFN '102
1110 M=S1/N: RETURN
```

В приведенном примере первое обращение к подпрограмме отличается от повторных обращений

местом входа, так как для накопления сумм необходимо инициализировать соответствующие переменные. Переменная S1 служит для хранения накопленных сумм, а в переменной N запоминается количество обращений к подпрограмме накопления сумм. Поэтому подпрограмма накопления сумм использует два входа, помеченные как 100 и 101 для соответственно первого и повторных обращений к подпрограмме. После первого обращения к подпрограмме значениям накопителя сумм (S1) и счетчика обращений присваивается значение нуля, а затем в программной строке 1060 происходит подсчет суммы элементов массива A(), а в программных строках 1070 и 1080 — увеличение счетчика числа обращений и накопителя сумм. При следующих обращениях к подпрограмме со входом 101 подсчитывается очередная сумма, увеличивается значение накопителя и счетчика обращений. При необходимости получения среднего значения накопленных сумм происходит обращение к подпрограмме со входом 102. Следует отметить, что при первом обращении к подпрограмме со входом 100 при ее выполнении в строке 1050 встречается оператор DEFFN' 101, который, как отмечалось ранее, не влияет на выполнение программы. Таким образом, в подпрограмме накопления сумм используются два входа и один оператор возврата.

Часто до обращения к подпрограмме необходимо ряду переменных присвоить определенные значения, аналогично тому, как при вычислении функций задаются значения аргументов. В следующем примере используется подпрограмма, вычисляющая остаток от деления целого числа A на целое число B. До обращения к подпрограмме необходимо задать значения A% и B%.

```
100  REM  ФОРМИРОВАНИЕ ОБРАЩЕНИЯ К ПОДПРОГРАММЕ
110  A%=110  :  B%  =  33
120  G0SUB  '50
. . .
300  REM  ПОДПРОГРАММА ВЫЧИСЛЕНИЯ ОСТАТКА ОТ
ДЕЛЕНИЯ
310  DEFFN  '50
320  M%=A-INT(A%/B%)*B%
330  RETURN
```

В результате выполнения подпрограммы переменной M% присваивается требуемое значение остатка от деления целого числа A на целое число B. Как видно из указанного примера, до перехода к подпрограмме следовало выполнить операторы присваивания переменным необходимых значений. В языке программирования машины предусмотрены удобные средства, позволяющие выполнять такие присваивания непосредственно в операторе перехода к подпрограмме. Для этого используются помеченные подпрограммы с параметрами.

Переменные подпрограммы, используемые в качестве параметров, записываются через запятую и помещаются в скобках в операторе помеченного входа в подпрограмму DEFFN'. В соответствующих обращениях к таким подпрограммам с использованием оператора GOSUB' в скобках помещаются разделенные запятыми фактические значения параметров, которые присваиваются переменным подпрограммам в момент выполнения оператора перехода к помеченной подпрограмме с параметрами. Так, подпрограмма предыдущего примера может принять следующий вид:

```
300  REM  ПОДПРОГРАММА ВЫЧИСЛЕНИЯ ОСТАТКА ОТ
ДЕЛЕНИЯ
310  DEFFN   '50(A%,   B%) 3 20 M%=A-INT(AVB%)*B%
330  RETURN
```

Обращение к этой подпрограмме может быть записано следующим образом:

```
500   GOSUB '50(110,33)
```

После выполнения подпрограммы переменная M% примет значения остатка от деления 110 на 33. Обращение к этой же подпрограмме может быть записано так:

```
500   E%  =  110:  C%=33:   GOSUB  '50(E%,   C%)
```

или

```
500  E% = 100: C % = 11:   GOSUB '50(E% + 10,   C%*3)
```

Таким образом, в качестве фактических параметров в операторе обращения к подпрограмме могут использоваться арифметические выражения.

В качестве формальных параметров подпрограммы, записываемых в операторе DEFFN', могут использоваться не только числовые, но и символьные переменные, которым при выполнении

соответствующих операторов перехода к подпрограмме могут присваиваться значения символьных переменных или констант.

Пример 10.7. Пример подпрограммы ввода числовых значений в символьную переменную с контролем количества вводимых разрядов и величины числа.

```
1000 REM ПОДПРОГРАММА ВВОДА ЧИСЛА С КОНТРОЛЕМ
1010 DEFFN '161(A⊙, K%)
1020 PRINT "ВВОД ЧИСЛА C"; K%; "РАЗРЯДАМИ, <=";A⊙
1030 INPUT C⊙
1040 REM ОПРЕДЕЛЕНИЕ КОЛИЧЕСТВА РАЗРЯДОВ В
ВВЕДЕННОМ ЧИСЛЕ
1045 P%=NUM(STR(C⊙, l, LEN(C⊙))
1050 IF P< = K% THEN 1070
1055 PRINT "СЛИШКОМ МНОГО РАЗРЯДОВ
1060 PRINT "ПОВТОРИТЕ ВВОД" : GOTO 1020
1070 B⊙=" "
1075 REM ВЫРАВНИВАНИЕ ВВЕДЕННОГО ЧИСЛА ДЛЯ
ПОСЛЕДУЮЩЕГО СРАВНЕНИЯ
1080 STR(B⊙, 1+K%-P%)=C⊙
1090 IF STR(B⊙, l, K%)>STR(A⊙, l, K%) THEN 1070
2000 RETURN
2100 PRINT "ЧИСЛО БОЛЬШЕ ДОПУСТИМОГО":GOTO 1060
```

При обращении к этой подпрограмме по оператору

```
GOSUB' 161("728",3)
```

в переменную C⊙ необходимо ввести символьное значение числа, не превышающего 728. При попытке ввода числа с большим количеством разрядов или превышающего заданную первым параметром величину выдается сообщение об ошибке и повторяется ввод. При обращении к подпрограмме

```
GOSUB' 161 ("992", 2)
```

в переменную C⊙ необходимо ввести символьное значение числа из двух разрядов, не превышающего 99.

Пример 10.8. Пример подпрограммы печати строки таблицы, образуемой из наименования изделия, планового и фактического количества изготавливаемых изделий и процента выполнения плана,

```
10 DIM A⊙(100), P(100), E(100)
. . .
100 REM ПОДПРОГРАММА ПЕЧАТИ СТРОКИ ВЫПОЛНЕНИЯ
ПЛАНА
110 DEFFN '40(C⊙, P, E)
120 REM ПРИ ОТСУТСТВИИ ПЛАНОВОГО ПОКАЗАТЕЛЯ
(P=0) ПРОЦЕНТ ВЫПОЛНЕНИЯ ПЛАНА ПРИНИМАЕТСЯ РАВНЫМ
100
125 IF P=0 THEN 140
130 REM ПРОЦЕНТ ВЫПОЛНЕНИЯ ПЛАНА ВЫВОДИТСЯ С
ТОЧНОСТЬЮ ДО ОДНОЙ ДЕСЯТОЙ
135 PRINT C⊙, P, E, ROUND(E*100/P,l);"%":GOTO
150
140 PRINT C⊙, " - ", E, 100;" % "
150 RETURN
```

Предположим, что в массивах A⊙ (), P() и E() записаны наименования, плановые и фактические количества изготавливаемых изделий. В этом случае вывод таблицы выполнения плана по К изделиям, начиная с К1-го, может выглядеть следующим образом:

```
1000 REM ПЕЧАТЬ ЗАГОЛОВКА ТАБЛИЦЫ
1005 PRINT "ТАБЛИЦА..."
1010 FOR C=K1 TO K1+K-1
1015 REM ПЕЧАТЬ СТРОКИ ТАБЛИЦЫ
1020 G0SUB ' 40(A⊙ (C), P(C), E(C))
1030 NEXT C
```

При обращении к подпрограммам с параметрами должны соблюдаться следующие правила:

количество фактических параметров, записанных в операторе GOSUB', должно соответствовать количеству формальных параметров, записанных в операторе DEFFN';

типы соответствующих фактических и формальных параметров должны быть одинаковыми. Недопустимо так же, как и в операторе присваивания, подставлять числовую переменную или константу в оператор GOSUB' на место, на котором в соответствующем DEFFN' стоит символьная переменная и, наоборот, подставлять символьную константу или переменную на место числовой переменной. Следующие обращения к описанным ранее примерам подпрограмм вызовут ошибку при исполнении операторов перехода к подпрограмме:

```
GOSUB'  50
GOSUB'  161(100,3)
GOSUB'  40 ("СТАНОК","—", 10)
```

10.5. Вызов подпрограмм с использованием клавиш специальных функций

На клавиатуре микроЭВМ вверху расположен ряд из 16 клавиш специальных функций, которые могут быть нажаты в верхнем или нижнем регистре клавиатуры. Эти клавиши могут определять 32 различные функции. В отличие от остальных клавиш клавиатуры, имеющих вполне определенные функции в различных режимах работы машины, клавиши специальных функций предназначены для выполнения действий, определяемых в программе, которая выполняется в данный момент. Иными словами, клавиши специальных функций применяются в качестве программируемых клавиш клавиатуры.

Клавиши специальных функций могут быть использованы для перехода к помеченным подпрограммам без параметров, номера которых находятся в диапазоне от 0 до 31, т. е. если подпрограмма помечена DEFFN'0, DEFFN'l и т. д. до DEFFN'31, то выполнение такой подпрограммы может быть начато нажатием клавиши специальных функций с соответствующим номером, например клавиши 0 для перехода к подпрограмме, начинающейся с оператора DEFFN'0. Использовать клавиши специальных функций для перехода к подпрограммам можно только после того, как записанная в памяти машины программа начала выполняться, т. е. она должна быть запущена по команде RUN. В дальнейшем, если выполнение подпрограммы было прервано из-за ошибок в программе или действий пользователя, переход к помеченным подпрограммам остается возможным до тех пор, пока не будет изменен текст программы.

Выполнение подпрограммы нажатием клавиши специальных функций может быть начато только в том случае, когда машина ожидает ввод информации с клавиатуры. Переход к подпрограммам с использованием клавиатуры специальных функций возможен только при выполнении оператора ввода INPUT (или L1NPUT), а также из режима непосредственного счета, но лишь в тех случаях, когда машина перешла в этот режим после нажатия клавиши RUN. Например, возможно начать выполнение подпрограммы нажатием клавиши специальных функций после выполнения оператора STOP.

Пример 10.9. Пусть в программе требуется в зависимости от требований пользователя производить различные арифметические действия (сложение, вычитание, деление и умножение) над двумя вводимыми числами. Для выполнения соответствующих операций используем подпрограммы, вызываемые нажатием клавиши специальных функций. Такая программа может выглядеть следующим образом:

```
100  REM  ИСПОЛЬЗОВАНИЕ КЛАВИШ СПЕЦИАЛЬНЫХ ФУНКЦИЙ
ДЛЯ ВЫПОЛНЕНИЯ РАЗЛИЧНЫХ ОПЕРАЦИЙ
110  PRINT  HEX(03)  :  REM ОЧИСТКА ЭКРАНА
120  PRINT,  "ДЕЙСТВИЯ",  "КЛАВИША СПЕЦИАЛЬНЫХ
ФУНКЦИЙ"
130  PRINT,  "A+B",  "0"  :  REM  ПЕЧАТЬ МЕНЮ
140  PRINT  "A-B",  "1"
I50 PRINT  "A  *  B",  "2"
160  PRINT  "A/B",  "3"
170  INPUT  "ВВЕДИТЕ ЧИСЛА  A  И  B", X,Y
180  STOP  "НАЖМИТЕ КЛАВИШУ НЕОБХОДИМОЙ ОПЕРАЦИИ"
190  GOTO  110
200  DEFFN  '0:  REM  ПОДПРОГРАММА СЛОЖЕНИЯ
210  PRINT  "РЕЗУЛЬТАТ  A+B="; X+Y
220  RETURN
```

```
230 DEFFN' 1: REM ПОДПРОГРАММА ВЫЧИТАНИЯ
240 PRINT "РЕЗУЛЬТАТ A-B/"; X-Y
250 RETURN
260 DEFFN '2: REM ПОДПРОГРАММА УМНОЖЕНИЯ
270 PRINT "РЕЗУЛЬТАТ A * B/"; X*Y
280 RETURN
290 DEFFN '3: REM ПОДПРОГРАММА ДЕЛЕНИЯ
300 IF Y=0 THEN 330
310 PRINT "РЕЗУЛЬТАТ A/B="; X/Y
320 RETURN
330 PRINT "ДЕЛЕНИЕ НЕВОЗМОЖНО: B=0"
340 RETURN
```

После вывода на экран меню, в котором поясняется соответствие номеров клавиш специальных функций и арифметических действий, программа переходит к вводу чисел, над которыми должны быть выполнены нужные арифметические операции. Затем в программе записан оператор останова, и после ввода чисел машина переходит в режим непосредственного счета, предварительно напечатав сообщение о необходимом действии пользователя. После нажатия пользователем соответствующей клавиши, например клавиши 3, машина ищет в тексте программы оператор DEFFN'3. Найдя его в строке 290, машина начинает выполнение этой подпрограммы. После выполнения операторов печати в строках 310 и 330 в зависимости от значения второго введенного числа выполняется оператор возврата. Так как выполнение подпрограммы начиналось в режиме непосредственного счета, после выполнения подпрограммы на экране в следующей печатной строке появится символ двоеточия и курсор, т. е. программа снова перейдет в режим непосредственного счета. Далее при нажатии любой из клавиш специальных функций от нуля до трех подпрограмма соответствующей арифметической операции выполнится снова и так далее, пока не будет нажата клавиша CONTINUE, после чего продолжается выполнение основной программы.

Выполнение подпрограмм нажатием клавиш специальных функций открывает широкий диапазон возможностей использования этих клавиш не только при выполнении, но и при отладке программ. Например, при отладке некоторой программы необходимо контролировать содержимое переменных M, K, P(1), P(3) и B⊙. Вместо того, чтобы всякий раз при останове отлаживаемой программы набирать с клавиатуры

```
PRINT M, K, P(1),P(3),B⊙
```

рекомендуется дополнить текст программы следующей подпрограммой:

```
3000 DEFFN'15
3010 PRINT M,K,P(1),P(3),B⊙
3020 RETURN
```

Теперь при необходимости печати значений соответствующих переменных достаточно одного нажатия клавиши 15 специальных функций в режиме непосредственного счета, и значения требуемых переменных будут распечатаны.

Помеченные подпрограммы, выполнение которых инициируется нажатием клавиш специальных функций, могут выполняться в течение длительного времени и не содержать операции вывода на экран машины. Чтобы информировать пользователя о работе подпрограммы, курсор от двоеточия переходит в первую позицию следующей строки экрана. После выполнения подпрограммы в этом месте высветится двоеточие.

Кроме обращения к помеченным подпрограммам в режиме непосредственного счета, удобно также использовать обращение к ним при выполнении оператора ввода с клавиатуры INPUT (или LINPUT). Выполнение подпрограммы при нажатии соответствующей клавиши происходит точно так же, как описано ранее, но по окончании подпрограммы происходит возврат к оператору INPUT (или LINPUT) основной программы, т. е. повторится выполнение операторов INPUT или LINPUT. Вызвать подпрограмму при выполнении оператора ввода можно в любой момент до нажатия клавиши CR/LF.

П р и м е р 10.10. Использование клавиши специальных функций для вызова подпрограмм, вычисляющих различные функции Десятичного логарифма, квадратного корня, экспоненты иллюстрирует следующая подпрограмма:

```
10 REM ВЫЗОВ ПОДПРОГРАММЫ НАЖАТИЕМ КЛАВИШИ
СПЕЦИАЛЬНЫХ ФУНКЦИЙ
```

```
20  PRINT  HEX(03)  :  REM ОЧИСТКА ЭКРАНА
30  PRINT  "ФУНКЦИИ",  "КЛАВИША СПЕЦИАЛЬНЫХ ФУНКЦИЙ"
40  PRINT  "LOG(X)",   "10" : REM  ПЕЧАТЬ МЕНЮ
50  PRINT  "LOG(X)",  "11"
60  PRINT  "LOG(X)",  "12"
70  INPUT  "ВВЕДИТЕ X И НАЖМИТЕ КЛАВИШУ НУЖНОЙ
ФУНКЦИИ", X
80  GOTO  20
90  DEFEN  10:  REM ВЫЧИСЛЕНИЕ ЛОГАРИФМА
100  PRINT  "L0G(";X;")=";L0G(X)/LOG(10)
110  RETURN
120  DEFEN  11:  REM ВЫЧИСЛЕНИЕ КВАДРАТНОГО КОРНЯ
130  PRINT  "SQR(";X;")=";SQR(X)
140  RETURN
150  DEFEN  12:REM ВЫЧИСЛЕНИЕ ЭКСПОНЕНТЫ
160  PRINT  "EXP(";X;")=";EXP(X)
170  RETURN
```

После вывода на экран меню с описанием клавиш специальных функций выполняется оператор ввода в строке 70. На экране высвечивается текст, записанный в операторе ввода, и появляется знак вопроса. После ввода с клавиатуры, например числа 100, и I нажатия клавиши специальных функций с номером 10 на экране появляется

```
LOG (100) =2
```

Затем снова выполняется оператор ввода в строке 70. Если на этот раз сразу же нажать клавишу специальных функций например клавишу 11, не вводя числовых значений, начинает выполняться подпрограмма вычисления квадратного корня. Так как значение X осталось прежним, т. е. равным 100, на экран выводится

```
SQR(100) = 10
```

Затем выполняется оператор возврата в программу из подпрограммы, программа переходит к вводу с клавиатуры. Если выполнить ввод и без нажатия клавиши специальных функций нажать клавишу CR/LF, выполнение программы продолжается с вывода меню.

Далеко не во всех случаях после обращения к подпрограмме требуется возврат к месту, откуда произошло обращение. Пусть, например, в начале программы содержится фрагмент, в котором вводятся данные, необходимые для выполнения программы. В процессе ввода этих данных иногда возникает необходимость вернуться в начало диалога. Для того чтобы этот возврат можно было осуществить из любого места, достаточно оформить начало диалоговой части в виде помеченной подпрограммы со специфической структурой. Она состоит из операторов DEFFN' и RETURN CLEAR. Оператор DEFFN' обеспечивает переход в начало диалога при нажатии соответствующей клавиши специальных функций, а оператор RETURN CLEAR — завершение подпрограммы без возврата в место вызова.

П р и м е р 10.11.

```
10  REM  ПРИМЕНЕНИЕ КЛАВИШ СПЕЦИАЛЬНЫХ ФУНКЦИЙ И
ОПЕРАТОРА  RETURN  CLEAR
20  GOSUB  '0  :  REM  ПЕРВОЕ ОБРАЩЕНИЕ К НАЧАЛУ ДИАЛОГА
30  REM  НАЧАЛО ДИАЛОГА
40  DEFFN  '0
50  RETURN  CLEAR:  REM  КОНЕЦ ВЫПОЛНЕНИЯ ПОДПРОГРАММЫ
60  INPUT  "ПАРАМЕТР  1",  A
70  INPUT  "ПАРАМЕТР  2",  B
. . .
200  INPUT  "ПАРАМЕТР  K",  K
210  REM  КОНЕЦ ДИАЛОГА И ПРОДОЛЖЕНИЕ ПРОГРАММЫ
```

Выполнение программы, фрагмент которой записан выше, начинается с первого обращения к подпрограмме, помеченной нулем, т. е. перехода к диалоговой части программы. Далее в программе используются операторы ввода необходимых параметров. Если при вводе очередного параметра оказывается, что необходимо повторить весь ввод сначала, достаточно нажать клавишу 0 специальных функций, и программа перейдет к выполнению подпрограммы. Так как первым оператором этой подпрограммы стоит оператор RETURN CLEAR, выполнение подпрограммы заканчивается без возврата, и далее выполняется строка 60 основной программы. После окончания всей процедуры ввода

выполнение программы продолжается. При любом нажатии клавиши 0 специальных функций при выполнении оператора ввода или в режиме непосредственного счета в других местах программы, ее выполнение будет снова начато со строки 60. Заметим, что первое выполнение диалоговой части подпрограммы начинается с оператора GOSUB '0 в строке 20. Если бы этой строки не было, то при выполнении программы после нажатия клавиши RUN возникала бы ошибка в строке 50, так как встретился бы оператор возврата без соответствующего обращения к подпрограмме.

10.6. Использование клавиш специальных функций для ввода текстов

Оператор DEFFN' с номером от нуля до 31 также может использоваться для задания текстовой константы. В этом случае нажатие определенной клавиши специальных функций вызовет печать текстовой константы, записанной в соответствующем операторе DEFFN'. Например, если в программе записан оператор

```
100  DEFFN'5  "ПОКАЗАТЕЛЬ"
```

нажатие клавиши специальных функций с номером 5 при выполнении оператора INPUT (LINPUT) или в режиме непосредственного счета приведет к тому, что символы ПОКАЗАТЕЛЬ станут частью вводимой строки текста. Надо отметить, что использование оператора DEFFN' для ввода текстовых констант не является переходом к выполнению подпрограммы и не затрагивает поэтому выполнение каких-либо операторов программы. Такое использование просто обеспечивает возможность одним нажатием клавиши сразу ввести нужный текст. Естественно, в частном случае этот текст может представлять собой число или часть числа.

П р и м е р 10.12. Пусть в программе требуется часто вводить; числа, начинающиеся со знаков "—100". Запишем в любом месте программы следующий оператор:

```
1000  DEFFN'  9"-100"
```

При вводе очередного числа после появления знака вопроса достаточно нажать клавишу специальных символов с номером 9, чтобы на экране появилось:

```
?—100
```

Если требуется ввести именно число —100, то теперь достаточно нажать клавишу CR/LF, а если требуется ввести число —1003, то достаточно ввести последнюю цифру 3 и нажать клавишу CR/LF. Таким же образом символы —100 могут быть введены в строку непосредственного счета.

Использование клавиш специальных функций может быть целесообразно также во время ввода и отладки программ. Так, если в программе часто используется оператор

```
GOSUB'122"
```

можно в режиме непосредственного счета определить десятую клавишу специальных функций

```
:DEFFN'10"GOSUB'122"
```

Теперь при необходимости ввода в строку программы текста этого оператора достаточно нажать клавишу специальных функций с номером 10 вместо последовательного нажатия пяти клавиш.

Если оператор DEFFN' с текстовой константой встречается при выполнении программы, то никаких действий по нему не производится. Оператор DEFFN' является неисполняемым оператором.

П р и м е р 10.13. Пусть при отладке программы необходимо время от времени распечатывать номера строк программы, содержащие обращения к подпрограмме с номером 56. Определим клавишу специальных функций 1

```
10  DEFFN'1"LIST 56"
```

Теперь после нажатия клавиши 1 в режиме непосредственного счета будет вводиться текст

```
:LIST  56
```

Для распечатки ссылок на подпрограмму 56 достаточно нажать клавишу CR/LF.

В заключение описания использования клавиш специальных функций для обращения к подпрограммам или ввода текстов необходимо отметить следующее:

нажатие клавиши специальных функций, не определенной оператором DEFFN', вызовет индикацию соответствующей ошибки;

при частом использовании клавиш специальных функций для выполнения соответствующих действий в каких-либо комплексах программ целесообразно делать смысловые надписи на планке с обозначением

номеров клавиш специальных функций. Это делается для того, чтобы пользователь готового комплекса программ мог использовать клавиши специальных функций по их смысловому обозначению аналогично остальным клавишам клавиатуры;

до использования клавиш специальных функций должна быть хотя бы один раз выполнена команда запуска программы RUN.

Глава 11. ОТЛАДКА И РЕДАКТИРОВАНИЕ ПРОГРАММ. ОБРАБОТКА ОШИБОК

Эта глава посвящена операторам языка Бейсик, предназначенным для отладки и редактирования текста программ. Отладкой программы называется процесс проверки правильности функционирования составленной программы. При отладке программы проверяется соответствие результатов ее работы результатам, которых ожидал программист, составивший программу. Ошибки, выявленные в процессе отладки, могут носить различный характер. Ошибки могут возникнуть из-за неправильно составленного алгоритма решения задачи или неправильно составленной программы. Ошибки могут возникать и вследствие неправильного использования каких-либо операторов или неправильной последовательности их выполнения. И наконец, могут быть допущены ошибки при вводе текста программы. В последнем случае диалоговый режим ввода программных строк в память машины позволяет машине контролировать правильное написание операторов, т. е. синтаксис вводимой строки проверяется машиной сразу же после ввода строки. Однако ошибки, связанные с пропуском отдельных операторов, набором неправильных наименований аргументов, неправильных номеров в операторах перехода и ряд других, могут пройти незамеченными при вводе текста и обнаружатся только во время выполнения программы.

Отладка является естественной и необходимой частью процесса составления программы. Без использования отладки, как правило, удается составлять только очень простые по структуре и небольшие по объему программы. Язык машины содержит мощные и разнообразные отладочные средства, позволяющие программисту быстро и эффективно проводить отладку программы в диалоговом режиме. Рассматриваемые далее отладочные средства языка машины могут использоваться не только при отладке программ. Ряд описываемых операторов может применяться в готовой отлаженной программе.

В этой же главе описываются операторы, позволяющие проводить программную обработку ошибок, возникающих при выполнении программ. Средства программной обработки ошибок позволяют создавать программы, сохраняющие программный контроль при возникновении сбоев оборудования или ошибок, вызванных неправильными действиями пользователя. При этом программа может продолжать выполнение, попытавшись устранить причины возникновения ошибок, обойти ошибку или, сообщив пользователю об ошибочной ситуации, ожидать его действий по устранению ошибки. В ряде случаев применение обработки ошибок в программе позволяет сократить и упростить программирование. Кроме того, использование программной обработки ошибок часто является единственным средством получения программой необходимой информации о состоянии внешних устройств в операциях ввода-вывода. Иначе любой сбой или отказ оборудования в операции ввода или вывода данных в программе без обработки ошибок приведет к останову выполнения программы и переходу машины после выдачи сообщения об ошибке в режим непосредственного счета. Программная обработка ошибок может использоваться также и при отладке программы.

11.1. Оператор останова программы и клавиша продолжения выполнения программы

Оператор останова программы STOP предназначен для останова программы и перевода машины в режим непосредственного счета. При выполнении этого оператора прекращается счет по программе, а на экране появляется сообщение —STOP. В следующей строке экрана высвечиваются двоеточие и курсор, что означает переход машины в режим непосредственного счета. Запись оператора STOP может сопровождаться символьной константой. В этом случае в тексте сообщения об останове программы за словом STOP выводится символьная константа. Если в операторе останова записан параметр #, после вывода символьной константы выводится номер программной строки, содержащей оператор останова. Оператор STOP имеет следующую форму записи:

STOP [<символьная константа>] [#]

После выполнения оператора останова машина переходит в режим непосредственного счета. Для

продолжения выполнения программы необходимо нажать клавишу CONTINUE, после чего выполнение программы продолжится со следующего за оператором STOP оператора в программе. Иногда дальнейшее выполнение программы не может быть продолжено из-за того, что в режиме непосредственного счета были выполнены следующие действия:

изменен текст программы;

нажата клавиша RESET;

введен необъявленный в программе массив;

выполнены операторы CLEAR V или CLEAR N;

произошло переполнение текстовой или табличной частей оперативной памяти машины (ошибки ERR 01 или ERR 02).

П р и м е р 11.1. В этом примере оператор останова используется для того, чтобы создать паузу при выполнении программы, во время которой пользователь мог бы привести необходимое для работы программы устройство (в данном случае печатающее устройство) в рабочее состояние,

```
100 REM ОПЕРАТОР STOP СЛУЖИТ ДЛЯ СОЗДАНИЯ
ПАУЗЫ ПРИ ВЫПОЛНЕНИИ ПРОГРАММЫ
110 INPUT "ВЫЧИСЛЕНИЕ ФАКТОРИАЛА ОТ P=1 ДО
P = ",N
120 STOP "ВКЛЮЧИТЕ ПЕЧАТАЮЩЕЕ УСТРОЙСТВО!"
130 SELECT PRINT OC
140 F=l
150 FOR P=1 TO N
160 F=F*P
170 PRINT "P="; P, "P!="; F
180 NEXT P
:RUN
ВЫЧИСЛЕНИЕ ФАКТОРИАЛА ОТ P = 1 ДО P = ?8 [CR/LF]
STOP ВКЛЮЧИТЕ ПЕЧАТАЮЩЕЕ УСТРОЙСТВО!
```

После появления этого сообщения пользователь может включить печатающее устройство, а затем продолжить выполнение программы, нажав клавишу CONTINUE.

Клавиша CONTINUE может также использоваться для продолжения выполнения программы в случае ошибки при выполнении оператора ввода или вывода, вызванной сбоем или неготовностью устройства к работе. Продолжать выполнение программы можно после устранения причины, вызвавшей ошибку.

П р и м е р 11.2. Вычеркнем из текста программы примера 11.1 строку с номером 120, т. е. уберем из текста программы оператор останова программы, и запустим программу на исполнение, отключив печатающее устройство. После запуска программы на экран будут выведены следующие сообщения:

```
:RUN
ВЫЧИСЛЕНИЕ ФАКТОРИАЛА ОТ P = 1 ДО P = ?10 [CR/LF]
170 PRINT "P=";P,"P!=";F
  ERR 8008
:__
```

Выполнение программы приостановлено из-за ошибки, вызванной попыткой вывода данных на невключенное печатающее устройство. После включения печатающего устройства и нажатия; клавиши CONTINUE машина продолжит выполнение программы.

При использовании оператора останова для создания пауз в выполнении программы утрачивается программный контроль нам действиями пользователя. А в режиме непосредственного счета, в который переходит машина после выполнения оператора останова, пользователь может нажимать любые клавиши на клавиатуре машины. При этом его действия могут вызвать одну из перечисленных ситуаций, после которой уже будет невозможно продолжить выполнение программы. Поэтому рекомендуется в текст сообщения в операторе STOP ввести указание пользователю программы, что он должен сделать для продолжения работы. Например

```
120 STOP «ВКЛЮЧИТЕ ПЕЧАТАЮЩЕЕ УСТРОЙСТВО! ДЛЯ
         ПРОДОЛЖЕНИЯ НАЖМИТЕ КЛАВИШУ CONTINUE»
```

Оператор STOP удобно использовать и при отладке программы. Пусть, например, определенная часть программы функционирует неправильно и известен номер программной строки, с которой начинается эта часть. Вставим на время оператор останова в начало этой строки для того, чтобы

остановить программу перед неправильно работающим местом. Затем с помощью средств, описанных в следующих разделах, можно в режиме пошагового выполнения программы определить и устранить причину ошибки. Например, при вставке в текст программы строки с номером

```
58  STOP  "СТРОКА"#
```

программа остановится при выполнении строки 58. После устранения ошибки из текста программы можно легко удалить данным отладочный оператор. Оператор останова удобно использовать описанным образом и в случае возникновения у программиста сомнения в том, что исполняется определенная ветвь программы, начинающаяся с известного номера строки, а также при необходимости проверки значений переменных в программе после выполнения определенной ее части. После выполнения оператора останова в этой части программы и перехода машины в режим непосредственного счета можно просмотреть значения требуемых переменных, а затем снова продолжить выполнение программы.

11.2. Непосредственное выполнение операций

Работа машины в режиме непосредственного счета уже рассматривалась в предыдущих главах. Этот режим также используется и при отладке программ. Дело в том, что в режиме непосредственного счета можно выполнять любые операторы языка Бейсик, изменяя, если нужно, значения переменных программы и не затрагивая при этом текст самой программы в памяти машины. Применение режима непосредственного счета для отладки программы иллюстрируется следующим примером.

П р и м е р 11.3. Данная программа предназначена для вычисления суммы первых N нечетных целых чисел. Она содержит ошибку, и сумма поэтому вычисляется неверно.

```
110  REM  ПРОГРАММА С ОШИБКОЙ ВЫЧИСЛЕНИЯ СУММЫ
120  PRINT  "ВЫЧИСЛЕНИЕ СУММЫ ПЕРВЫХ N НЕЧЕТНЫХ
ЧИСЕЛ"
130  PRINT  "ВВЕДИТЕ N  ",N
140  S=l
150  IF  N=1  THEN  200
160  FOR  K=l  TO  N
170  D=D+2
180  S=S+D
190  NEXT  K
200  PRINT  "СУММА  N  =  ";S
```

Если в качестве N вводится число 5, то эта программа вычисляет **сумму,** равную 21. Правильный ответ должен равняться 25, так как сумма первых нечетных N чисел равна N^2. Для поиска ошибки в выполнении программы в строку 185 вставляем оператор останова и выполняем программу

```
185  STOP  "СТРОКА  "  #
:RUN
ВЫЧИСЛЕНИЕ СУММЫ ПЕРВЫХ N НЕЧЕТНЫХ ЧИСЕЛ
ВВЕДИТЕ  N  ?  5[CR/LF]
STOP  СТРОКА  185
:_
```

Теперь в режиме непосредственного счета можно проверить значения переменных D и S, распечатав их содержимое:

```
:PRINT   "D = ";D;"S = ";S
D = 2   S = 3
:__
```

Распечатав значения D и S, убеждаемся в том, что ошибка в расчете уже существует. В строке 170 переменная D должна приобретать нечетные значения, а ее значение в данный момент четное. Продолжим выполнение программы, еще раз пройдя цикл вычисления следующего нечетного числа и суммы, а после останова программы снова распечатаем значения D и S:

```
STOP СТРОКА  185
:PRINT "D = ";D;"S = ";S
D=4   S=7
:__
```

Теперь понятно, что цикл получения следующего нечетного числа и суммы (текст программы в строках 160—190) составлен правильно, а ошибка состоит в том, что до входа в цикл значение D равнялось нулю, т. е. четному числу, а не 1.

Проверим наше предположение, присвоив в режиме непосредственного счета переменным D и S значение единицы, а затем перейдем к выполнению программы со строки 160, которая является началом цикла подсчета суммы

```
:D,S=1
:GOTO   160
:__
```

Продолжим в этом месте выполнение программы нажатием клавиши CONTINUE. Выполнение программы окончится в строке 185. Распечатаем затем значения переменных D и S:

```
STOP  CTPOKA   185
:PRINT   "D = ";D;"S = ";S
D = 3   S=4
:__
```

Нажав клавишу CONTINUE, пройдем еще раз цикл вычислений, и снова напечатаем значения переменных D и S:

```
STOP  CTPOKA   185
:PRINT   "D = ";S = ";S
D = 5   S = 9
:__
```

Убеждаемся, что в программе ошибки нет. Изменим строку 140 так, чтобы присвоить переменной D значение единицы, и удалим строку с номером 185. После выполнения этих операций получим правильно работающую программу

```
110  REM  ОТЛАЖЕННАЯ ПРОГРАММА
120  PRINT  "ВЫЧИСЛЕНИЕ СУММЫ ПЕРВЫХ  N  НЕЧЕТНЫХ
ЧИСЕЛ"
130  PRINT  "ВВЕДИТЕ  N  ",N
140  S,D=1
150   IF  N=1   THEN   200
160  FOR  K=l   TO  N
170  D=D+2
180  S=S+D
190  NEXT  K
200  PRINT "СУММА N   =  ";S
```

Чтобы начать теперь выполнение программы, необходимо использовать оператор RUN, так как нажатие клавиши CONTINUE после исправления текста программы приведет к ошибке.

11.3. Пошаговое выполнение программы

Клавиша HALT/STEP предназначена для пошагового выполнения программы. Нажатие клавиши HALT/STEP приводит к останову программы после исполняемого в момент нажатия оператора программы. Если, например, в момент нажатия клавиши HALT/STEP выполняется операция ввода с клавиатуры, то останов происходит после окончания ввода данных с клавиатуры, после нажатия клавиши CR/LF. После нажатия клавиши HALT/STEP в начале следующей строки экрана высвечиваются двоеточие и курсор. Машина переходит в режим непосредственного счета точно так же, как и при выполнении оператора останова. Таким образом, одной из функций клавиши HALT/STEP является останов выполнения программы с клавиатуры в любой момент времени. Для такого останова использование клавиши HALT/STEP предпочтительнее использования клавиши сброса RESET, так как после останова программы по клавише HALT/STEP выполнение программы может быть потом продолжено нажатием клавиши CONTINUE.

Первое нажатие клавиши HALT/STEP останавливает выполнение программы. При повторном нажатии этой клавиши (или после ее первого нажатия при останове программы по оператору STOP) выполняется следующий оператор программы. Затем исполнение программы прекращается, машина

снова переходит в режим непосредственного счета. В таком режиме выполнения программы по одному оператору — пошаговом режиме—может, быть пройдена часть программы или вся программа.

При нажатии клавиши HALT/STEP на экран выводится номер программной строки и текст этой строки. Первый высвечиваемый оператор этой строки уже выполнен. Очередное нажатие клавиши HALT/STEP приводит к появлению на экране номера строки, следующего выполненного оператора в этой строке и следующих за ним операторов в строке. После выполнения и высвета последнего оператора в строке очередное нажатие клавиши HALT/STEP приводит к выводу содержимого следующей программной строки и исполнению первого оператора в этой строке.

Таким образом, последовательное нажатие клавиши HALT/STEP позволяет исполнять программу шаг за шагом, высвечивая текущие операторы программы. Пошаговый режим выполнения бывает очень полезен при поиске и устранении ошибок в программе.

Пример 11.4. Используем в этом примере программу примера 11.3, содержащую ошибку вычисления суммы, с добавлением оператора останова в строке 185.

```
110   REM  ПРОГРАММА С ОШИБКОЙ ВЫЧИСЛЕНИЯ СУММЫ
120   PRINT  "ВЫЧИСЛЕНИЕ СУММЫ ПЕРВЫХ N НЕЧЕТНЫХ
ЧИСЕЛ"
130   INPUT  "ВВЕДИТЕ N  ",N
140   S=l
150   IF N=1  THEN  200
160   FOR  K=1  TO  N
170   D=D+2
180   S=S+D
185   STOP  "СТРОКА"  #
190   NEXT  K
200   PRINT  "СУММА  N = ";S
```

Выполнение программы выглядит следующим образом:

```
:RUN
ВЫЧИСЛЕНИЕ СУММЫ ПЕРВЫХ N НЕЧЕТНЫХ ЧИСЕЛ
ВВЕДИТЕ N  ?  5 [CR/LF]
STOP  СТРОКА  185
:__
```

При нажатии клавиши HALT/STEP на экране появляется сначала номер строки 185, в которой произошел останов, и при последующем нажатии высвечивается следующая программная строка:

```
185
:
190  NEXT   K
:__
```

Последовательное нажатие клавиши HALT/STEP приведет к выводу на экран дисплея следующей информации:

```
170   D = D+2
:
180  S = S+D
:
185  STOP СТРОКА #
STOP"СТРОКА"185
:__
```

Программа, выполняемая в пошаговом режиме, выполняется так же, как и работающая непрерывно. Нажатием клавиши CONTINUE в любой момент можно перейти к непрерывному выполнению программы.

11.4. Отладочный режим выполнения программы

Наряду с описанными средствами отладки программы в машине предусмотрен и специальный отладочный режим выполнения программы. В этом режиме при выполнении операторов, изменяющих содержимое любых переменных программы, а также при выполнении операторов, нарушающих обычный порядок выполнения программы, на экран выводится соответствующая информация. Переход в отладочный режим осуществляется с помощью оператора TRACE.

В отладочном режиме каждый раз, когда при выполнении программы переменной присваивается новое значение, на экран выдается соответствующее сообщение. Сообщение об изменении содержимого переменной имеет вид:

<идентификатор переменной> = <присваиваемое значение>

Каждый раз, когда в программе происходит переход к строке по операторам GOTO, GOSUB, IF-THEN и подобным им, на экране высвечивается сообщение о переходе на соответствующую программную строку и номер этой строки в виде

TRANSFER TO<номер строки перехода>

При выполнении оператора конца цикла высвечивается следующее сообщение:

NEXT<переменная цикла> = <новое значение переменной цикла>

Это сообщение выводится каждый раз при новом исполнении цикла. Если цикл завершен, высвечивается сообщение

NEXT END<переменная цикла>

Машина переходит в отладочный режим при выполнении оператора TRACE в режиме непосредственного счета или в результате выполнения оператора TRACE, записанного в тексте программы. Отладочный режим сохраняется в машине, пока не будут выполнены оператор TRACE OFF (оператор отмены отладочного режима) или оператор очистки памяти CLEAR.

При использовании в отладочном режиме пошагового выполнения программы становится возможным проследить весь ход выполнения какой-либо части программы и достаточно быстро обнаружить ошибки в работе программы.

Пример 11.5. В этом примере используем программу примера 11.4, содержащую ошибку в подсчете суммы первых нечетных чисел.

Выполнение программы выглядит следующим образом:

```
:RUN
ВЫЧИСЛЕНИЕ СУММЫ ПЕРВЫХ N НЕЧЕТНЫХ ЧИСЕЛ
ВВЕДИТЕ N? 5[CR/LF]
STOP СТРОКА 185
:TRACE [CR/LF]
:
```

После останова программы по оператору STOP в строке программы 185 вводим с клавиатуры и выполняем оператор TRACE в режиме непосредственного счета. Теперь машина находится в отладочном режиме. После последовательного нажатия клавиши пошагового выполнения на экране высветится следующая информация:

```
 185
 :
 190  NEXT K
 NEXT K=2
```

При выполнении оператора конца цикла на экран выводится значение переменной цикла, а затем выполняются следующие операторы за оператором начала цикла (сообщения о переходе в этом случае не возникает). Последовательное нажатие клавиши пошагового выполнения приведет к следующей индикации:

```
:
170 D = D+2
D= 4
180 S=S+D
S= 7
:
185 STOP "СТРОКА" #
STOP СТРОКА 185
:
185
:
190 NEXT K
NEXT K= 3
170 D=D+2
D= 6
180 S=S+D
```

```
S=  13;
:
185  STOP  "СТРОКА"  #
STOP  СТРОКА  185
:
```

В этом месте, выполнив нужную часть программы, выключаем отладочный режим, набрав с клавиатуры следующий оператор;

```
:TRACE   OFF   [CR/LF]
```

Далее нажатием клавиши CONTINUE переводим машину в режим обычного выполнения программы со следующего за оператором останова оператором.

Использование клавиши пошагового выполнения программы в отладочном режиме позволяет визуально наблюдать за выполнением отлаживаемой программы. Однако в ряде случаев при отладке программы достаточно использовать только отладочный режим без останова выполнения программы или ее отлаживаемой части. Проиллюстрируем это на примере приведенной программы (отлаженной программы примера 11.3), удалив из программы оператор останова в строке 185. До исполнения программы включим отладочный режим, выполнив

```
:TRACE
```

После запуска программы получим следующие результаты:

```
:RUN
ВЫЧИСЛЕНИЕ СУММЫ ПЕРВЫХ N НЕЧЕТНЫХ ЧИСЕЛ
ВВЕДИТЕ N ? 5 [CR/LF]
S= 1
D= 1
TRANSFER  TO  200
СУММА  =  1
:__
```

или

```
:RUN
ВЫЧИСЛЕНИЕ СУММЫ ПЕРВЫХ N НЕЧЕТНЫХ ЧИСЕЛ
ВВЕДИТЕ N ? 5 [CR/LF]
S= 1
D= 1
K= 2
D= 3
S= 4
NEXT  K= 3
D= 5
S= 9
NEXT K= 4
D= 7
S= 16
NEXT  END K
СУММА = 16
:_
```

В отладочном режиме пользователь может, просматривая значения изменяемых переменных программы, легко найти причину неправильной работы. Однако при большом объеме индицируемых данных следить за выполнением программы при выводе на экран машины с обычной скоростью будет трудно. После заполнения всего экрана его содержимое сдвинется на одну строку, и далее данные на экране будут мелькать со скоростью, превышающей возможности нашего восприятия. Следовательно, необходимы средства уменьшения скорости вывода информации на экран. Язык машины предоставляет возможность введения временных задержек после выполнения каждой операции вывода строки. Для этого используется оператор SELECT с параметром в виде

```
SELECT  P<цифра>
```

Оператор SELECT P задает паузу после каждой выводимой строки. Продолжительность этой паузы кратна 1/6 секунды. Например, после выполнения оператора

```
:SELECT P1
```

каждая строка предыдущего примера будет выводиться на экран через 1/6 секунды.

Длительность паузы может быть увеличена при задании в операторе цифры, большей 1. Можно задавать цифры от 1 до 9. Например, после выполнения оператора

```
:SELECT P6
```

продолжительность пауз между выводом строк будет составлять одну секунду, а после выполнения оператора

```
:SELECT P9
```

интервал времени между выводом строк достигнет максимального значения 1,5 секунды.

После того как задан режим выполнения с паузами, задержка при выводе строк на экран будет происходить до тех пор, пока не будет выполнен другой оператор SELECT P без параметра, т. е.

```
:SELECT P
```

Оператор SELECT P может быть использован не только при отладке программ, но и для задания пауз при выводе информации на экран дисплея и на другие устройства вывода, например на печатающее устройство.

11.5. Индикация текста, переменных и переходов программы и таблицы устройств

Как уже известно, для вывода текста программы, записанной в оперативной памяти машины, используется оператор LIST. Различные формы этого оператора позволяют выводить на экран дисплея или печатающее устройство не только текст программы, но и номера программных строк, содержащих определенные переменные, номера программных строк, содержащих обращения к заданным строкам, а также текущее состояние таблицы устройств машины.

Во всех формах оператора LIST можно использовать в качестве параметра номер устройства вывода, который задается в виде косой черты и двух шестнадцатеричных цифр адреса устройства.

Оператор LIST с параметром * позволяет находить и выводить строки программы, содержащие заданный в операторе LIST* текст. Общая форма этого оператора следующая:

```
LIST [S] [<устройство>,]*<текст> [<номер строки 1>] [,<номер строки 2>]
```

где текст задается символьной константой или переменной, а номера строк определяют диапазон строк программы, внутри которых ведется поиск заданного текста.

Рассмотрим результат выполнения оператора

```
:LIST*"PRINT"
```

при загруженной в память машины программе примера 11.4. В результате на экран будут выведены две строки программы, содержащие слово «PRINT»:

```
120 PRINT "ВЫЧИСЛЕНИЕ СУММЫ ПЕРВЫХ N НЕЧЕТНЫХ
ЧИСЕЛ"
200 PRINT "СУММА N = "; S
```

Оператор LIST* удобно применять в самых различных случаях контекстного поиска по программе. Например, хотя и существует рассматриваемая ниже форма оператора LIST V для поиска переменных в тексте программы, часто бывает необходимо найти только определенный элемент массива. В этом случае удобнее использовать оператор LIST*, чем просматривать все найденные оператором LIST V строки. Пусть, например, необходимо найти, в каких программных строках используются элементы второй строки массива B(). Для вывода этих строк используем оператор

```
:LIST*"B(2,"
```

Для поиска элемента массива B(3,8) используем оператор

```
:LIST*"B(3,8"
```

Для поиска в тексте программы символа кавычек надо выполнить оператор

```
:LIST*/"/
```

В заключение описания оператора LIST* отметим, что в задаваемом для поиска тексте концевые пробелы не входят в искомый текст.

Еще одна форма оператора LIST с параметром V служит для вывода номеров программных строк, в которых содержится заданная переменная или несколько переменных. Общая форма оператора LIST V в этом случае выглядит следующим образом:

LIST V [S] [<устройство>,] [<имя переменной 1>][,<имя переменной 2>]

где в качестве имени переменной могут использоваться имя целой, действительной или символьной переменной, а также обозначение целого, действительного или символьного массивов. Для обозначения массива используются имя массива, открывающие и закрывающие скобки. Например,

```
LIST V A( )
```

При выполнении оператора LIST V выводятся имя переменной, дефис и номера всех программных строк, где встречается эта переменная. Если в операторе опущены остальные параметры, то в таком виде индицируются списки строк, содержащих все переменные программы. Сначала распечатываются все строки, содержащие целые переменные, затем действительные, символьные, потом обозначения целых массивов, обозначения действительных массивов и обозначения символьных массивов. Вывод каждого типа переменных происходит в алфавитном порядке, начиная с имени A и кончая именем Z9, если, конечно, такие переменные есть в программе.

Пусть, например, в память машины введен текст программ примера 11.4, а затем выполнен оператор LIST V. На экран выводится следующая информация:

```
:LIST V
D—0170 0180
K—0160 0190
N—0130 0150 0160
S—0140 0180 0200
```

Таким образом, использование оператора LIST имеет чрезвычайно важное значение при отладке программы, так как позволяет выводить список переменных и строки программы, где они используются. При дополнении или изменении текста программе использование оператора LIST V позволяет вводить имена новым переменных, избегая повторного использования каких-либо переменных. А при необходимости изменения уже имеющейся в программе переменной оператор LIST V позволяет вывести все строки программы, в которых есть эта переменная.

Для задания диапазона выводимого списка переменных используются параметры оператора LIST V.

П р и м е р 11.6. Пусть в оперативную память машины введена следующая программа:

```
10 DIM D9⊙(3),C⊙ (2,3),A⊙ (8),N(40),B9%(2)
20 A1=5.5:X,X1,Y=0.3:F3%,W%=100
30 N(l)=Al*Y/2
40 B⊙=D9⊙*(l)
50 B9%(2)=F3%*Y
```

Оператор LIST V обеспечивает вывод списка всех переменных программы в виде

```
:LIST V
F3% - 0020  0050
W% - 0020
A1 - 0020  0030
X - 0020
X1 - 0020
Y - 0020  0030  0050
B - 0040
B9%( ) - 0010  0050
N( ) - 0010  0030
A⊙( ) - 0010
S⊙( ) - 0010
D9⊙( ) - 0010  0040
```

Оператор LIST V с одним параметром — именем переменной 1 распечатывает все номера строк программы, содержащих эту переменную. Например

```
:LIST V A1
A1-0020 0030
:LIST V W%
W%-0020
:LIST V D9⊙( )
D9⊙( )-0010 0040
```

Если в тексте программы нет переменной с именем, используемой в операторе LIST V, то выполнение этого оператора заканчивается без индикации, например

```
:LIST V Z
:—
```

При использовании в операторе LIST V второго параметра— имени переменной 2 распечатываются все имена имеющихся переменных в алфавитном порядке в диапазоне от имени переменной 1 до имени переменной 2. Например

```
:LIST V X,Y
X-0020
X1-0020
Y-0020 0030 0050
:LIST V A%,X%
F3%-0020 0050
W%-0020
:LIST  V  A⊙( ),D⊙( )
A⊙( ) - 0010
S⊙( ) - 0010
```

Еще одна форма оператора LIST используется для индикации перекрестных ссылок к строкам с указанными номерами. Общая форма этого оператора имеет следующий вид:

```
LIST⊙[S] [<устройство>,] [<номер строки 1>]
[,<номер строки 2>]
```

При выполнении оператора LIST 0 выводятся номер программной строки, дефис и номера всех программных строк, в которых записаны операторы перехода на данную строку (или другие операторы, использующие в качестве параметра номер данной строки). Если в операторе LIST© не используются параметры, то в таком виде индицируются все строки программы, на которые есть ссылки из других строк программы. При этом, конечно, возможны и ссылки из одной строки на самое себя, например

```
10  GOTO   10
```

П р и м е р 11.7. Пусть в оперативной памяти машины находится следующая программа:

```
10  GOTO  100
20  A=A+1:IFA>10 THEN 30:PRINT :G0SUB 200
30  PRINT  2*A
40  GOTO  20
100  IF  A=0 THEN  20:A=0:GOTO  40
200  PRINT  "ЧИСЛО=";:RETURN
:LIST
0020 - 0040  0100
0030 - 0020
0040 - 0100
0100 - 0010
0200 - 0020
```

Если в операторе LIST⊙ задан один параметр — номер строки 1, то при выполнении оператора печатаются все строки программы, на которые есть ссылки, начиная с заданной строки, например

```
:LIST⊙  40
0040 - 0100
0100 - 0010
0200 - 0020
```

При использовании в операторе LIST⊙ обоих параметров выводятся номера строк с ссылками на номера в диапазоне строк, начиная с номера строки 1, и кончая номером строки 2. Например

```
:LIST⊙ 20,20
0020 — 0040 0100
:LIST⊙ 20,40
0020 — 0040 0100
0030 — 0020
0040 — 0100
```

При отладке программы удобно, используя оператор вывода перекрестных ссылок, определять

правильность переходов в программе. Иначе приходилось бы для такого определения просматривать весь текст программы, а такой просмотр при большом объеме программы был бы весьма трудоемким занятием.

Оператор LIST с параметром % предназначен для вывода текущего состояния таблицы устройств машины. Эта форма оператора имеет вид

```
LIST [<устройство>,]   %
```

Таблица устройств выводится в следующем виде:

```
:LIST%
CI = 01   CO = 05(80)   PRINT=05(80)   LIST=05(80)   TAPE = 08(256)
PLOT=10
#0—18F
#1–
#2–
#3–
#4–
#5–
#6–
#7–
```

где

CI и CO — соответственно устройства консольного ввода (клавиатура) и консольного вывода (экран);
PRINT — устройство вывода для операторов PRINTUSING, HEXPRINT и MATPRINT;
LIST — устройство вывода для операторов LIST;
TAPE — устройство ввода и вывода для операторов DATA LOAD BT и DATA SAVE BT;
PLOT — устройство вывода для графических операторов.

В скобках за физическим адресом устройства выводится текущая длина строки устройства вывода. В строчках с символами # и номерами от 0 до 7 выводится информация о текущем состоянии файлов. В качестве такой информации выводятся номера устройств, а для файла на диске еще и номера начального, текущего и конечного секторов файла на диске. Эта информация выводится, если в процессе работы программы был использован файл с соответствующим номером. Автоматически для файла с номером 0 определен дисковод I8F.

Использование оператора LIST % позволяет в любое время получить информацию о состоянии таблицы устройств машины. Пусть, например, выполнен оператор

```
:SELECT  PRINT  0C(132)
```

Распечатав таблицу устройств, пользователь может узнать текущее назначение устройства вывода для операторов печати.

```
CI = 01   CO = 05(80)   PRINT = 0C(132)   LIST = 05(80)   TAPE = 08(256)
PLOT=10
#0—18F
#1–
#2–
#3–
#4–
#5–
#6–
#7–
```

11.6. Перенумерация и стирание программных строк и переменных программы

При вводе текста программы номера строк могут вводиться с помощью клавиши номера строки на клавиатуре или непосредственным набором цифр номера строки. Однако часто в процессе составления программы приходится вставлять целые фрагменты программы в какую-либо ее часть, а эти фрагменты могут состоять из большого количества строк. Чтобы вставить в программу новые строки, не заменяя имеющихся в программе строк, нужно новым вводимым строкам задавать номера, которых нет в программе. А если надо вставить больше девяти программных строк между строками 0010 и 0020 программы или хотя бы одну строку между строками 0136 и 0137 программы, следует перенумеровать все или часть строк исходной программы. Такая перенумерация в принципе возможна, так как

абсолютные значения номеров программных строк сами по себе ничего не означают. Они являются просто метками операторов, расположенных вместе в одной строке, для операторов перехода, ссылающихся на данную строку из других строк программы. Поэтому можно перенумеровать программу или ее часть, заменив одни номера программных строя другими и сохранив естественный порядок возрастания номеров при замене во всех операторах, ссылающихся на программные строки, прежних номеров на новые. При такой перенумерации по лучим программу, совершенно идентичную исходной, но с новыми значениями номеров программных строк. В языке машины для подобной перенумерации предусмотрен специальный оператор RENUMBER.

Оператор перенумерации без параметров перенумеровывает целиком весь текст исходной программы. При этом в получения программе новые номера строк программы начинаются с номера 0010 и имеют значения, кратные десяти.

П р и м е р 11.8. В памяти машины до перенумерации был записана следующая программа:

```
10  INPUT K
12  IF INT(K/2)= K/2 THEN 15
13  PRINT "ЧИСЛО НЕЧЕТНОЕ"
14  GOTO 16
15  PRINT "ЧИСЛО ЧЕТНОЕ"
16  STOP
```

Выполним в режиме непосредственного счета перенумерацию и выведем текст программы

```
:RENUMBER
:LIST
10  INPUT K
20  IF INT(K/2)=  K/2  THEN  50
30  PRINT  "ЧИСЛО НЕЧЕТНОЕ"
40  GOTO  60
50  PRINT  "ЧИСЛО ЧЕТНОЕ"
60  STOP
```

Полученная в результате перенумерации программа идентична исходной программе с точностью до номеров строк, т. е. выполнение обеих программ приведет к абсолютно одинаковым результатам. Перенумерация коснулась не только номеров программных строк, в которых записаны операторы, но и всех номеров программных строк, используемых в операторах. Так, в операторе условного перехода IF — THEN номер строки перехода 15 поменялся на 50, в операторе перехода GOTO номер строки поменялся с 16 на 60.

Оператор перенумерации может использоваться и с параметрами, задающими процесс перенумерации. Общая форма оператора перенумерации имеет следующий вид:

RENUMBER [<номер строки 1>][,<номер строки 2>][,<шаг>]

где

<номер строки 1> — номер первой строки в программе, начиная с которой строки программы будут перенумерованы. Строки программы с номерами меньше <номера строки 1> после нумерации будут иметь прежние номера. Конечно, номера строк в самих операторах при этом могут быть перенумерованы. Если параметр <номер строки 1> пропущен, перенумеровываются все строки программы;

<номер строки 2> — новый номер строки, соответствующей первой перенумеруемой строке программы. Если этот параметр опущен, в качестве нового номера первой перенумерованной строки используется значение третьего параметра, т. е. шага перенумерации;

<шаг> — шаг перенумерации, определяющий, с каким шагом будут записываться новые значения номеров. Значение этого параметра может находиться в диапазоне от 1 до 99. По умолчанию значение шага принимается равным 10.

Перенумерация в предыдущем примере была проведена оператором RENUMBER без параметров, т. е. в этом случае значения параметров перенумерации были заданы по умолчанию. Теперь рассмотрим другие способы перенумерации программы примера 11.8.

Перенумеруем исходную программу с 10-й строки с новым номером 200:

```
:RENUMBER 10,200
:LIST
200  INPUT  K
210  IF INT(K/2)=  K/2 THEN 240
220  PRINT  "ЧИСЛО НЕЧЕТНОЕ"
230  GOTO  250
240  PRINT "ЧИСЛО ЧЕТНОЕ"
```

```
250  STOP
```

Полученная в результате перенумерации программа начинается с номера 200, а остальные ее строки записаны с шагом 10,1 К такому же результату привело бы исполнение оператора

```
:  RENUMBER,  200
```

где запятая означает, что первый параметр опущен.

В следующем примере используются все три параметра для перенумерации части программных строк примера 11.8, начиная со строки 13 с новым номером, равным 15, и шагом перенумерации, равным 2.

```
:RENUMBER 13,15,2
:LIST
10  INPUT  K
12  IF  INT(K/2)=  K/2 T HEN  19,
15  PRINT  "ЧИСЛО НЕЧЕТНОЕ"
17  GOTO  21
19  PRINT  "ЧИСЛО ЧЕТНОЕ"
21  STOP
```

В последнем примере в качестве нового номера строки задан имеющийся номер в перенумеруемой программе. Такое возможно. Однако, если перенумеровывается только часть программы, новый номер строки не может быть меньше номера строки, записанного в первом параметре, т. е. нельзя в данном случае задать

```
:RENUMBER  13,  10
```

Как уже говорилось, для очистки всей оперативной памяти машины от программ и данных следует использовать оператор CLEAR. Однако при редактировании текста программы часто бывает необходимо удалитъ из памяти только часть содержащихся там программных строк. Сделать это можно различными способами. Во-первых, набирая при вводе с клавиатуры номер имеющейся программной строки, можно записать вводимую строку под тем же номером. При этом прежняя строка будет заменена на вводимую. Например, если в машину введен текст программы предыдущего примера

```
10  INPUT K
12  IF INT(K/2)= K/2 THEN 19
15  PRINT "ЧИСЛО НЕЧЕТНОЕ"
17  GOTO 21
19  PRINT "ЧИСЛО ЧЕТНОЕ"
21  STOP
```

и далее в режиме ввода набрать

```
:17  PRINT  "НОВОЕ  ЧИСЛО":  GOTO   10 :
21   GOTO   10
:__
```

а затем вывести измененный текст программы, то строки 17 и 21 окажутся замененными на введенные.

```
:LIST
10   INPUT  K
12   IF  INT(K/2)=  K/2 THEN  19
15   PRINT  "ЧИСЛО НЕЧЕТНОЕ"
17   PRINT  "НОВОЕ ЧИСЛО":GOTO 10
19   PRINT  "ЧИСЛО ЧЕТНОЕ"
21   GOTO  10
```

Отдельную строку из текста программы можно удалить не заменяя ее на новую. Для этого нужно набрать номер удаляемой строки, например 17, и вслед за этим нажать клавишу возврата

```
:LIST
10  INPUT  K
12  IF  INT(K/2)=  K/2 THEN  19
15  PRINT  "ЧИСЛО  НЕЧЕТНОЕ"
19  PRINT  "ЧИСЛО ЧЕТНОЕ"
21  GOTO  10
```

Однако, если из текста программы нужно удалить некоторый Диапазон программных строк, удалять каждую строку этого диапазона предыдущим способом будет слишком трудоемко.

Для удаления из памяти ЭВМ части текста программы используется оператор CLEAR с параметром P. Оператор CLEAR P удаляет часть или целиком весь текст программы. Общая форма этого оператора следующая:

```
CLEAR P   [<номер строки 1>]  [,<номер строка 2>]
```

где

<номер строки 1>—номер строки, начиная с которого удаляются строки программы. Если этот параметр опущен, то текст программы стирается с первой строки;
<номер строки 2> — номер последней стираемой строки. Если этот параметр опушен, текст программы стирается до конца.

Например, после выполнения следующего оператора:

```
CLEAR  P  40,  180
```

в тексте программы не будет строк, начиная со строки с номером 40 и кончая строкой с номером 180.

Пусть в памяти машины записан текст программы предыдущего примера. После выполнения оператора CLEAR P в памяти останется следующий текст:

```
:CLEAR P 15,21
:LIST
10 INPUT K
12 IF INT(K/2)= K/2 THEN 19
```

К такому же результату привело бы выполнение оператора

```
:CLEAR  P  15
```

Оператор CLEAR P может быть использован и без параметров. Выполнение оператора CLEAR P без параметров не аналогично выполнению оператора CLEAR. При выполнении оператора CLEAR P из памяти машины стирается только текст программы, и никаких других изменений не происходит. Остаются неизменными значения переменных в программе, текущие значения таблицы устройств, назначение устройства по оператору SELECT и т. п.

Для очистки памяти машины от значений переменных и тех переменных, которых нет в настоящее время в тексте программы, используется оператор CLEAR с параметрами V или N. Текст программы при этом остается неизменным, а все переменные, используемые в программе, получают исходные значения (пробелы для символьных переменных и нули для числовых). Операторы CLEAR V и CLEAR N не аналогичны друг другу. После выполнения оператора CLEAR V стираются значения всех переменных, и в том числе общих переменных, заданных оператором COM. Оператор CLEAR N стирает значения только необщих переменных. Рассмотрим следующий пример:

```
10  DIM  A⊙1,  B⊙1
20  A⊙ = "А":B⊙ = "БВГДЕ":M = 2
30  PRINT  A⊙;B⊙;M
:RUN
АБ 2
```

При выполнении программы выводится содержимое переменных A⊙ длиной в один символ, B⊙ длиной в один символ и числовое значение переменной M. Так как длина переменной B⊙ —один байт, то ей присваивается в строке 20 программы только первый символ константы «БВГДЕ». Изменим содержимое строки 10 программы, удалив из нее определение переменной B⊙

```
10 DIM  A⊙1
```

и далее выполним оператор печати

```
:PRINT  A⊙  ;  B⊙;  M
АБ  2
```

Значения, распечатываемые предыдущим оператором, остались, естественно, без изменения. Выполним далее операторы:

```
:CLEAR  V
:B⊙ = "0123456789АБВГДЕ"
:PRINT  A⊙;B⊙;M
0123456789АБВГДЕ  0
```

После выполнения оператора CLEAR V переменная B⊙ была удалена из памяти. Новое

использование этой переменной приводит к иным результатам. Длина переменной В⊙ теперь стала равна шестнадцати символам, а переменные А⊙ и М получили начальное значение. Введем в память машины и выполним программу примера 12.9, а затем выполним следующие операторы в режиме непосредственного счета:

```
:CLEAR P
:LIST
/\ERR 23
:PRINT A⊙;B⊙;M
АБ 2
:_
```

В памяти машины текст программы отсутствует (ошибка с кодом 23 обозначает отсутствие программы в памяти ЭВМ), а переменные А⊙, В⊙ и М сохраняют свои значения.

11.7. Оператор конца программы

В языке машины предусмотрен оператор обозначения конца программы. Им является оператор END. Использование этого оператора в конце записи текста программы не обязательно, так как выполнение программы автоматически заканчивается после выполнения последней ее строки. В то же время оператор END можно вставлять в любые строки программы, когда необходимо остановить ее выполнение. На экране появляется сообщение

```
END PROGRAM
FREE SPACE XXXXX
```

где XXXXX — пятиразрядное число, равное количеству свободных байтов памяти.

Назначением оператора END, кроме останова счета по программе, является индикация текущего объема свободной области оперативной памяти машины.

В памяти машины хранятся как тексты программы, так и значения переменных. Область памяти, отводимая под переменные, может изменяться при выполнении операторов DIM, а область памяти, занятая текстом программы, изменяется при загрузке отдельных частей программы с диска. Поэтому объем свободной памяти машины изменяется во время исполнения программы. При отладке программы, в которой используется много переменных с большой размерностью, можно легко выйти за границы допустимого размера памяти. Для того чтобы контролировать оставшееся незанятым место памяти, можно использовать оператор END в режиме непосредственного счета. После вывода сообщения о размере свободной области памяти можно решать вопрос о допустимости увеличения размерности переменных, увеличения числа переменных или увеличения текста программы.

11.8. Ошибки в программе и их обработка

На этапе выполнения программы могут возникать различные ошибки, причины которых — сбои или отказы оборудования машины, неправильные действия пользователя, переполнение памяти машины и т. п. При возникновении ошибки средства диагностики машины определяют ее характер, выдают сообщение об ошибке и переводят машину в режим непосредственного счета. Все диагностируемые машиной ошибки подразделяются на две группы: системные ошибки, возникающие из-за сбоя процессора или памяти машины, и программные ошибки, возникающие при выполнении программы. Сообщение о системной ошибке выводится в следующем виде: (СИСТЕМЫ)

$$\text{СБОЙ}\left\{\begin{matrix}\text{СИСТЕМЫ}\\ \text{МО}\\ \text{ОЗУ}\\ \text{УП}\end{matrix}\right\}$$

Причиной такой ошибки может быть сбой процессора, оперативной или управляющей памяти машины. Так как системная ошибка может привести к искажению записанных в памяти машины данных, попытка продолжения выполнения программы может привести к непредсказуемым результатам. В общем случае после возникновения системных ошибок следует заново привести машину в рабочее состояние.

При возникновении программной ошибки на экран дисплея выводится сообщение в виде

/\ERR XX

где XX —код (номер) ошибки.

Если ошибка произошла при выполнении операций ввода или вывода (кроме ошибок, вызванных неправильным вводом данных по оператору INPUT), сообщение о ней выдается в виде

```
/\ERR  80  XX
```

где XX —код ошибки операции ввода-вывода,

При возникновении ошибки в выполняемой программе перед строкой с сообщением об ошибке высвечивается часть строки программы, начинающаяся с оператора, в котором произошла ошибка. Сообщению об ошибке предшествует символ «/\», указывающий на строку с ошибкой,

Пусть, например, при выполнении программы встретилась строка с неправильным оператором присваивания:

```
100 A⊙=1
/\ERR   12
:__
```

При исполнении этой строки возникнет ошибка с кодом 12 (неправильный оператор), выполнение программы прекратится. Подобную ошибку можно оперативно исправить, например, введя

```
:100 A⊙= "1"
:RUN   100
```

После этого программа продолжит свою работу.

Ошибка, возникающая при исполнении оператора INPUT из-за неправильного ввода данных, в отличие от других ошибок не приводит к прекращению выполнения программы. При этом на экране высвечивается сообщение об ошибке и ввод данных повторяется. Пусть, например, при выполнении строки программы

```
. . .
500 INPUT  "ЧИСЛО",  C
. . .
```

пользователь вводит с клавиатуры следующие данные:

```
 ЧИСЛО?  A   [CR/LF]
 /\ERR  29
 ?21   [CR/LF]
```

Ошибка при первоначальном вводе возникла из-за попытки ввода текстового значения вместо нужного здесь числового значения. После индикации сообщения машиной и повторного ввода пользователем правильного значения программа продолжит работу.

Ряд ошибок, возникающих при выполнении программы, носит вычислительный характер. Это происходит тогда, когда ошибка вызвана недопустимым значением какой-либо числовой переменной, возникшим в результате вычисления арифметического выражения. Одной из таких ошибок является математическая ошибка (ошибка переполнения), возникающая, например, при попытке деления на нуль, Пусть, например, в память машины введена программа вычисления частного от двух вводимых с клавиатуры чисел

```
10  INPUT  "ДЕЛИМОЕ,  ДЕЛИТЕЛЬ", A,B
20  PRINT  "ЧАСТНОЕ = ",  A/B
```

Если во время работы этой программы при вводе второго числа задать нуль, то при выполнении программной строки с номером 20 возникнет ошибка

```
ДЕЛИМОЕ,  ДЕЛИТЕЛЬ?  21,0
20 PRINT  "ЧАСТНОЕ", A/B
/\ERR  03
:__
```

Ошибка с кодом 03 означает математическую ошибку. Ошибка ввода-вывода может возникнуть и при работе правильной программы, например, в результате обращения к неготовому внешнему устройству. Допустим, в программе используется вывод на печатающее устройство, и до запуска программы печатающее устройство не было включено. При выполнении оператора печати возникнет ошибка

```
150   PRINT/0G,  "РЕЗУЛЬТАТ=";  T
/\ERR  8008
:__
```

Ошибка с кодом 8008 означает аварию устройства ввода или вывода. Выполнение программы при возникновении вычислительных ошибок и ошибок в операциях ввода-вывода данных приостанавливается на операторе, в котором возникла ошибка. Если причина возникновения этих ошибок будет устранена, то выполнение программы может быть продолжено. Например, в двух предыдущих примерах ошибки могли быть устранены следующим образом. При выполнении программы вычисления частного после останова программы по ошибке введем в режиме непосредственного счета отличное от нуля значение переменной В. Далее можно продолжить выполнение программы нажатием клавиши CONTINUE:

```
ДЕЛИМОЕ, ДЕЛИТЕЛЬ?  21,0
20 PRINT  "ЧАСТНОЕ", A/B
/\ERR  03
:B=7   [CR/LF]
: [CONTINUE]
ЧАСТНОЕ=3
```

При работе программы, выводящей данные на печатающее устройство, после возникновения сообщения об ошибке, приведем печатающее устройство в состояние готовности и продолжим выполнение программы нажатием клавиши продолжения работы программы

```
150  PRINT/OC,  "РЕЗУЛЬТАТ=";T
/\ERR  8008
: [CONTINUE]
```

Ошибки, которые при устранении причин, их вызывающих, не мешают продолжению работы программы, называются программообрабатываемыми. Не все ошибки являются программообрабатываемыми. После возникновения ошибок, вызванных переполнением памяти машины текстом или переменными программы, ошибок ввода программных сегментов с диска и подобных им продолжить выполнение программы невозможно.

Для программной обработки ошибок в языке предусмотрен специальный оператор обработки ошибок. Этот оператор срабатывает в программе только тогда, когда возникает программообрабатываемая ошибка. Оператор обработки ошибок имеет следующий вид:

```
ON  ERROR        [<символьная        <символьная        GOTO <номер
                  переменная 1>,     переменная 2>           строки>]
```

При возникновении ошибки в программе в <символьную переменную 1> заносится код ошибки (два или четыре символа). В <символьную переменную 2> заносится четырехзначный номер программной строки, в которой произошла ошибка. Далее в программе осуществляется переход по указанному в операторе номеру строки. Для того чтобы ошибка была обработана подобным образом, оператор ON ERROR-GOTO должен встретиться по ходу выполнения программы ранее оператора, при выполнении которого произошла ошибка.

П р и м е р 11.10. Составим программу, вычисляющую частное от деления двух вводимых чисел и выдающую сообщение при ошибке переполнения результата.

```
10  DIM E⊙4,K⊙4
20  ON ERROR E⊙,K⊙ GOTO 70
30  INPUT "ДЕЛИМОЕ", A
40  INPUT "ДЕЛИТЕЛЬ", B
50  PRINT "ЧАСТН0Е=", A/B
60  STOP
70  PRINT "ДЕЛИТЕЛЬ СЛИШКОМ МАЛ !":60T0 40
```

Результат работы программы:

```
:RUN
ДЕЛИМОЕ ? 100
ДЕЛИТЕЛЬ ?   0
ДЕЛИТЕЛЬ СЛИШКОМ МАЛ !
ДЕЛИТЕЛЬ? 20
ЧАСТНОЕ= 5
STOP
:_
```

Выполнение оператора ON ERROR-GOTO без параметров отменяет программную обработку ошибок. После выполнения в программе такого оператора любая дальнейшая ошибка вызовет

сообщение об ошибке и останов работы машины. В одной и той же программе может использоваться неограниченное количество операторов ON ERROR-GOTO с различными параметрами и номерами строк перехода к обработке ошибок. Обработка возникшей ошибки всегда при этом проводится с учетом параметров последнего выполненного оператора обработки ошибок.

Существуют еще две формы оператора обработки ошибок, в которых вместо параметра GOTO используются параметры THEN или GOSUB. Операторы ON ERROR-THEN и ON ERROR-GOSUB отличаются от оператора ON ERROR-GOTO тем, что при возникновении обрабатываемой по этим операторам ошибки осуществляется переход к подпрограмме обработки ошибок. Подпрограмма обработки ошибок начинается с номера строки перехода, указанного в операторе обработки ошибок. Отличие операторов перехода к подпрограмме обработки ошибок состоит в следующем. При выполнении оператора возврата RETURN в подпрограмме обработки ошибок происходит возврат к месту возникновения ошибки. При этом, если используется оператор ON ERROR-THEN, при возврате происходит переход в программе к оператору, следующему за тем оператором, в котором произошла ошибка. А при использовании оператора ON ERROR-GOSUB происходит переход в программе к оператору, при выполнении которого произошла ошибка.

Пример 11.11. Пусть в программе требуется пропускать операторы, в которых могут возникать математические ошибки, и останавливать работу программы при возникновении других ошибок. Используем для обработки ошибок следующую подпрограмму:

```
10  DIM E04,K04
20  ON ERROR E⊙,K⊙ THEN 1000
. . .
ТЕКСТ ОСНОВНОЙ ПРОГРАММЫ
. . .
990  REM ПОДПРОГРАММА ОБРАБОТКИ ОШИБОК
1000  IF E⊙ <> "03" THEN 1010: RETURN
1010  PRINT "ОШИБКА ";E⊙;"В СТРОКЕ ";K⊙
1020  ON ERROR
1030  STOP
```

При возникновении ошибки в каком-либо операторе программы происходит переход к строке 1000, с которой начинается подпрограмма обработки ошибок. Сначала проверяется код ошибки. Если ошибка математическая, то выполняется оператор возврата RETURN и происходит возврат в основную программу. При этом оператор в основной программе, на котором произошла ошибка, пропускается, и программа переходит к выполнению следующего за ним оператора. Если ошибка вызвана другими причинами, выводится сообщение о коде ошибки и номере строки, где она произошла. Обработка ошибок отменяется, и выполнение программы прекращается оператором останова в строке 1030.

Пример 11.12. В программе в строках с 500 по 600 результаты выводятся на печатающее устройство. Для вывода сообщения пользователю программы о необходимости подготовки печатающего устройства к работе используем следующую подпрограмму обработки ошибок:

```
500  REM ВЫВОД НА ПЕЧАТАЮЩЕЕ УСТРОЙСТВО
510  SELECT PRINT OC:ON ERROR A⊙,B⊙ GOSUB 800
. . .
ПЕЧАТЬ РЕЗУЛЬТАТОВ
. . .
600  SELECT PRINT 05
. . .
790  REM ПОДПРОГРАММА ОБРАБОТКИ ОШИБОК
800  SELECT PRINT 05
810  PRINT HEX(07);:ST0P "ПОДГОТОВЬТЕ ПЕЧАТЬ"
820  SELECT PRINT OC:RETURN
```

Во время работы программы при выводе на печать может возникнуть ошибка из-за неподготовленности печатающего устройства. По заданному в строке 510 условию обработки ошибок будет выполняться подпрограмма, начинающаяся со строки 800. После вывода звукового сигнала и сообщения на экран выполнение подпрограммы прекращается. Устранив причину неисправности, пользователь должен продолжить выполнение подпрограммы нажатием клавиши CONTINUE. Далее происходит возврат в основную программу к оператору печати, на котором произошла ошибка, и осуществляется печать выводимых данных.

Часть III. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ЯЗЫКА МАШИНЫ

Глава 12. РАБОТА С ТАБЛИЦАМИ

12.1. Обозначения и соглашения о размерностях

Матричные операторы позволяют выполнять действия над числовыми массивами (матрицами и векторами) в соответствии с правилами линейной алгебры, а также обрабатывать (печатать, вводить значения элементов и т. п.) символьные массивы. Краткие сведения о матричных операторах приводятся в табл. 12.1.

Синтаксис матричных операторов во многом отличается от синтаксиса других операторов Бейсика. Каждый матричный оператор должен начинаться со слова MAT. При записи матричных операторов массивы обозначаются без круглых скобок после имени массива. Так, буквы A и B в матричном операторе MAT A = B означают ссылку на числовые массивы A и B, а не на простые переменные A и B.

Таблица 12.1

Операция	Описание	Пример записи
MAT PRINT[3]	Печать элементов массива	MAT PRINT A
MAT INPUT[1;3]	Ввод элементов массива с клавиатуры	MAT INPUT A
MAT READ[1;3]	Массив = значения, заданные в операторе DATA	MAT READ B,C
MAT равенство[2]	Матрица=матрица	MAT B = **C**
MAT сложение[2]	Матрица=матрица+матрица	MAT A=B+C
MAT вычитание[2]	Матрица=матрица—матрица	MAT A=B—C
MAT умножение[2] на скаляр	Матрица = скаляр * матрица	MAT A=(3)*A
MAT умножение[2]	Матрица = матрица * матрица	MAT A=B*C
MAT обращение[1] и детерминант	Матрица = обратная матрица, переменная = определитель матрицы	MAT A=INV(B)
MAT CON	Каждый элемент матрицы=1	MAT B = CON
MAT ZER[1]	Каждый элемент матрицы = 0	MAT A=ZER
MAT IDN	Матрица=единичная матрица	MAT A=IDN
MAT транспонирование[2]	Массив = транспонированный массив	MAT A=TRN(B)
MAT REDIM[1;3]	Переопределение размерности массива	MAT REDIM A(15,10)

[1] Размерность результирующего массива переопределяется явно (т. е. указывается новая размерность);
[2] Размерность результирующего массива переопределяется неявно (т. е, зависит от размерностей массивов — аргументов);
[3] Операция может выполняться с символьными массивами.

Обозначение массивов	
в матричных операторах	в других операторах
A	A( )
C2	C2( )
A⊙	A⊙ ( )
E1⊙	E1⊙ ( )
K4%	K4% ( )

Если в матричном операторе употребляется имя массива, который не был описан с помощью оператора DIM или COM, то тем самым определяется массив размерностью 10 строк и 10 столбцов. Так, например, программа, состоящая из одной строки

```
20  MAT A = B+C
```

эквивалентна следующей программе

```
10  DIM A(10, 10),  B(10, 10),  C(10, 10)
20  MAT A=B+C
```

Символьный массив, не описанный в операторе DIM или COM и участвующий в матричном операторе, автоматически объявляется размерностью 10*10 элементов длиной 16 символов каждый.

Ряд матричных операторов может изменять размерности массивов таким образом, чтобы объем памяти, занимаемый массивом с измененными размерностями, не превосходил объема, первоначально выделенного под этот массив. Например, оператор MAT REDIM специально используется для этой цели,

```
10  DIM R(10,15)
.
.
.
```

```
500 MAT  REDIM  R(8,X)
```

При выполнении такой программы после команды RUN машина выделит оперативную память для хранения 150 элементов массива R. Затем в строке 500 массив R переопределяется с новыми размерностями; теперь число строк в нем равно 8, а число столбцов равно значению целой части переменной X при условии, что 8*X≤150. Если это условие не выполняется (т. е. X>18), то при выполнении строки 500 произойдет останов по ошибке. При работе с символьными массивами можно изменять длину элемента символьного массива:

```
10  DIM   A⊙ (20,20) 20
.
.
.
500  MAT   REDIM  AO (25,30) 10
```

Первоначально под массив AO выделяется 20*20*20 = = 8000 байт ОЗУ. В строке 500 увеличивается число элементов A⊙ от 400 до 750 за счет уменьшения длины элемента массива с 20 до 10 байт. После выполнения строки 500 можно снова изменить размерность A⊙. Например,

```
700  MAT   REDIM   AO(10,20) 40
```

В этом случае общее число байтов, занимаемое массивом A⊙ (), снова станет равным 8000.

Пять матричных операторов позволяют переопределять размерности явно, так, как это делается при использовании оператора MAT REDIM. Кроме того, еще 6 матричных операторов автоматически определяют размерности результирующего массива в зависимости от размерностей массивов, участвующих в операции.

Существующие несколько ограничений при использовании матричных операторов сводятся к следующему. Одномерный массив не может быть переопределен как двумерный, и наоборот. Нельзя выполнять несколько действий в одном MAT-операторе, например MAT A = B + C — E. В этом случае следует использовать два оператора:

```
MAT A=B+C
MAT A=A-E
```

В операторах транспонирования и умножения матриц имя одной и той же матрицы не должно появляться по обе стороны знака равенства, например

```
MAT C=C*B
MAT C=TRN(C)
```

12.2. Матричные операторы

12.2.1. ОПЕРАТОР MAT PRINT (МАТРИЧНАЯ ПЕЧАТЬ)

Общая форма:

MAT PRINT [<устройство>], <имя массива> [{ , / . / , } [<имя массива>]] ...

Оператор предназначен для печати содержимого числовых и символьных массивов в соответствии с тем порядком, в каком они указаны в операторе. Каждый массив печатается строка за строкой. Первый элемент каждой строки массива печатается с новой строки. Если в операторе MAT PRINT за именем массива стоит запятая, элементы массива печатаются в зонном формате; если точка с запятой, в плотном формате. Для символьных массивов размер зоны устанавливается равным максимальной длине элемента. Одномерные массивы печатаются в виде столбца.

Если указан параметр <устройство>, то печать осуществляется на этом устройстве, в противном случае — на устройстве, определенном последним выполненным оператором SELECT PRINT, например MAT PRINT/OC, A%, B.

Пример 12.1

```
10  DIM A(3,3), B(3)
20  MAT'A=C0N: B(2)=-5
30  MAT PRINT A,B
40  MAT PRINT /05,A;
```

В результате выполнения программы будет напечатано:

а) содержимое массива A в зонном формате

```
 1          1          1
 1          1          1
 1          1          1
```

б) содержимое одномерного массива B

```
 0
-5
 0
```

в) содержимое массива A в плотном формате

```
 1   1   1
 1   1   1
 1   1   1
```

12.2.2. ОПЕРАТОР MAT INPUT (МАТРИЧНЫЙ ВВОД)

Общая форма:

$$\text{MAT INPUT}\begin{cases}<\text{имя числового массива}>\left[(<\text{а.в.1}>\left[,<\text{а.в.2}>\right])\right]_{...}\\ <\text{имя символьного массива}>\left[(<\text{а.в.1}>\left[,<\text{а.в.2}>\right])\right.\\ \left.\left[<\text{а.в.3}>\right]\right]...\end{cases}$$

где <а.в.1>, <а.в.2>—выражения, определяющие новые размерности;
<а.в.3> — выражение, определяющее максимальный размер каждого элемента символьного массива.

Оператор предназначен для ввода элементов числовых или символьных массивов с клавиатуры. Во время исполнения оператора MAT INPUT на экран выводится знак вопроса, и машина ожидает ввод данных. Элементы массива вводятся последовательно строка за строкой и разделяются запятыми. Символьные элементы с начальными пробелами или запятыми необходимо при вводе заключать в кавычки. Если при вводе возникла ошибка, его нужно повторить, начиная с ошибочно введенного элемента. Данные, введенные до момента возникновения ошибки, сохраняются.

Отсутствие данных во входной строке (нажатие клавиши CR/LF после знака вопроса) дает машине указание прекратить ввод данных в текущей строке, остальные элементы массивов в списке аргументов оператора MAT INPUT не учитываются.

Пример 12.2

```
10 DIM  A(2),  B(3),  C(3,,4)
10 MAT   INPUT  A,B(2),C(2,4)
```

Выполним эту программу:

```
? 1,2,3   [CR/LF] — вводятся элементы A(1), A(2),  B(1).  На экране снова возникает знак  вопроса, так как список  ввода
                    не  исчерпан;
?4,5,-6   [CR/LF] — заполняются элементы B(2), C(1,1), C(1,2)
?[CR/LF]          — ввод заканчивается, оставшиеся элементы массива C —C(1,3),  C(1,4),  C(2,1),  C(2,2),  C.(2,3),
                    C(2,4)   не изменяются.
```

12.2.3. ОПЕРАТОР MAT READ (МАТРИЧНОЕ ЧТЕНИЕ ДАННЫХ)

Общая форма:

$$\text{MAT READ}\begin{cases}<\text{имя числового массива}>\left[(<\text{а.в.1}>\left[,<\text{а.в.2}>\right])\right]_{...}\\ <\text{имя символьного массива}>\left[(<\text{а.в.1}>\left[,<\text{а.в.2}>\right])\right.\\ \left.\left[<\text{а.в.3}>\right]\right]...\end{cases}$$

где

<а.в.1>,<а.в.2> — выражения, определяющие новые размерности;
<а.в.3> — выражение, определяющее максимальную длину элемента символьного массива.

Оператор предназначен для присваивания элементам указанных массивов значений из списка аргументов оператора DATA. Массивы, указанные в операторе MAT READ, заполняются последовательно строка за строкой. Данные выбираются из списка значений операторов DATA в порядке возрастания номеров строк, содержащих оператор DATA. Если данных, определенных в операторах DATA, недостаточно для заполнения массива и других операторов DATA в программе нет, то происходит останов по; ошибке.

Пример 12.3

```
10  DIM    A(2),    B(3,3)
20  MAT    READ    A,B(2,3)
```

```
. . . . . .
50  MAT   PRINT   A, B
. . . . . .
100  DATA   1,  -2.3,  -3,6.2,  0.0,  1,  -5
```

В результате выполнения программы будет напечатано:

а) значения массива A

```
1
-2.3
```

б) значения массива B, преобразованного к новой размерности 2*3 в результате выполнения строки 20

```
-.3     6.2     0
 0       1     -5
```

12.2.4. ОПЕРАТОР MAT= (МАТРИЧНОЕ ПРИСВАИВАНИЕ)

Общая форма:

```
MAT  A=B
```

где

A, B — имена числовых массивов,

Оператор предназначен для присваивания значения элемента массива A соответствующему элементу массива B. Размерность массива A изменяется в соответствии с размерностью массива B.

Пример 12.4

```
10  MAT  A=B
20  MAT  B2%=C%
```

12.2.5. ОПЕРАТОР MAT+ (МАТРИЧНОЕ СЛОЖЕНИЕ)

Общая форма:

```
MAT C=A+B
```

где C, A и B — имена числовых массивов.

Оператор предназначен для сложения двух числовых массивов одинаковой размерности. Значение элемента матрицы C равно сумме значений соответствующих элементов матриц A и B. Размерность результирующего массива C изменяется на размерность массива A (или B). Если размерности массивов A и B не совпадают, то машина выдает сообщение об ошибке.

Пример 12.5

```
10  DIM  A(5,5),  P(5,5),  E%(7),  F%(5),  C%(5)
20  MAT  A = A + P
30  MAT  E% = F% + C%
40  MAT  A = A + A
```

Отметим, что в результате выполнения строки с номером 30 размерность массива E станет равной 5.

12.2.6. ОПЕРАТОР MAT - (МАТРИЧНОЕ ВЫЧИТАНИЕ)

Общая форма:

```
MAT C=A—B
```

где A, B и C — имена числовых массивов.

Оператор предназначен для вычитания матриц одинаковой размерности. Значение каждого элемента матрицы C равно разности значений соответствующих элементов матриц A и B. Размерность матрицы C переопределяется в соответствии с размерностью матрицы A (или B). Матрица C может стоять по обе стороны от знака равенства.

Если матрицы A и B имеют разные размерности, то происходит останов по ошибке.

Пример 12.6

```
10  DIM  A(5,5),  P(5,5),  E%(7),  F%(5),  C%(5)
20  MAT  A = A — P
30  MAT  E% = F% — C%
```

12.2.7. ОПЕРАТОР MAT( )* (УМНОЖЕНИЕ МАТРИЦЫ НА СКАЛЯР)

Общая форма:

```
MAT  C=(K)*A
```

где C и A—имена числовых массивов, K—выражение.

Оператор предназначен для умножения каждого элемента матрицы A на выражение K. Результат помещается в соответствующий элемент матрицы C. Матрица C может стоять по обе стороны знака равенства.

Пример 12.7

```
10  MAT  X=(5)*X
20  MAT  C = .(3*T+2)*B
```

В результате выполнения оператора умножения матрицы на число размерность матрицы-результата C переопределяется в соответствии с размерностью массива A.

12.2.8. ОПЕРАТОР MAT* (УМНОЖЕНИЕ МАТРИЦ)

Общая форма:

```
MAT  C=A*B
```

где

C, A и B — имена числовых массивов.

Оператор предназначен для умножения двух матриц. Матрицы A и B умножаются по правилам линейной алгебры, результат записывается в матрице C. Матрица C не должна появляться по обе стороны от знака равенства. Число строк матрицы A должно быть равно числу строк матрицы B (в противном случае происходит останов по ошибке). Размерность матрицы C определяется числом строк матрицы A и числом столбцов матрицы B.

Пример 12.8

```
10  DIM  A(3,3),  B(3,3),  C(3,3)
20  MAT  INPUT  B,  C
30  MAT  A = B*C
```

Ниже приведен результат на примере умножения матриц B и C с конкретными значениями:

$$B = \begin{Vmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 1 & 0 & 0 \end{Vmatrix} \qquad C = \begin{Vmatrix} 1 & 0 & 1 \\ 3 & 1 & 0 \\ 1 & 1 & 1 \end{Vmatrix} \qquad A = \begin{Vmatrix} 10 & 5 & 4 \\ 5 & 3 & 2 \\ 1 & 0 & 1 \end{Vmatrix}$$

12.2.9. ОПЕРАТОР MAT INV (ОБРАЩЕНИЕ МАТРИЦЫ, ВЫЧИСЛЕНИЕ ОПРЕДЕЛИТЕЛЯ МАТРИЦЫ)

Общая форма:

```
MAT C=  INV(A)  [,D]
```

где

A,C — имена числовых массивов;

D—имя числовой переменной, которой присваивается значение определителя матрицы A.

Оператор предназначен для вычисления матрицы C, обратной для матрицы A. Матрица A может появляться по обе стороны знака равенства, например

```
MAT A=INV(A)
```

Размерность матрицы C изменяется на размерность матрицы A. Матрица A должна быть квадратной и невырожденной, т. е. определитель ее не должен быть равен нулю. В противном случае выполнение программы прекращается по ошибке.

Если при записи оператора обращения указана переменная D, то после выполнения этого оператора D содержит значение определителя матрицы A, например

```
20  MAT  R = INV(B), B1
40  MAT  C = INV(C), X(3)
```

Пример 12.9. Пусть необходимо решить систему n линейных уравнений с n неизвестными:

$$C_{11}\cdot X_1 + C_{12}\cdot X_2 + \ldots + C_{1n}\cdot X_n = B_1$$
$$C_{21}\cdot X_1 + C_{22}\cdot X_2 + \ldots + C_{2n}\cdot X_n = B_2$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$C_{n1}\cdot X_1 + C_{n2}\cdot X_2 + \ldots + C_{nn}\cdot X_n = B_n$$

где

$X_1, X_2, \ldots, X_n$ — неизвестные;

$C_{11}, \ldots, C_{nn}$ — коэффициенты при неизвестных;

$B_1, \ldots, B_n$— правые части уравнений.

Эта система может быть записана в матричных обозначениях

$C \cdot X = B$,

где

C — квадратная матрица коэффициентов при неизвестных размером *n*n;*
X — вектор неизвестных;
B — вектор правых частей.

Решение этого матричного уравнения можно записать в следующем виде:

$X = C^{-1} \cdot B$,

где

C^{-1} — обратная матрица для матрицы C.

С помощью матричных операторов решение можно получить в два этапа: сначала получить обратную матрицу для матрицы коэффициентов C, а затем вычислить результат — вектор X. Ниже приводится текст программы для решения этой системы, причем для ввода известных величин используется оператор MAT INPUT, для печати — оператор MAT PRINT.

```
100   REM РЕШЕНИЕ СИСТЕМЫ. ЛИНЕЙНЫХ УРАВНЕНИЙ
110   DIM  C(10,10)  : REM  МАТРИЦА КОЭФФИЦИЕНТОВ
120   DIM X(10)       : REM  ВЕКТОР НЕИЗВЕСТНЫХ
130   DIM B(10)       : REM  ВЕКТОР ПРАВЫХ ЧАСТЕЙ
140   PRINT HEX (03)
150   REM  ПЕРЕОПРЕДЕЛЕНИЕ РАЗМЕРНОСТЕЙ МАССИВОВ
160   REM  ЕСЛИ ПОРЯДОК СИСТЕМЫ МЕНЬШЕ 10
170   INPUT  "ВВЕДИТЕ ЧИСЛО ПЕРЕМЕННЫХ  (Y<=10)",Y
180   MAT  REDIM  C(Y,Y), X(Y), B(Y)
190   REM  ВВОД МАТРИЦЫ КОЭФФИЦИЕНТОВ C
200   PRINT  HEX(03)  "ВВЕДИТЕ МАТРИЦУ КОЭФФИЦИЕНТОВ"
210   PRINT  "СТРОКА ЗА СТРОКОЙ, РАЗДЕЛЯЯ ЭЛЕМЕНТЫ"
220   PRINT  "ЗАПЯТЫМИ. ПОСЛЕ ВВОДА ОЧЕРЕДНОЙ СТРОКИ
230   PRINT  НАЖИМАЙТЕ КЛАВИШУ.CR/LF"
240   MAT  INPUT  C
24 5   REM  ВВОД ВЕКТОРА ПРАВЫХ ЧАСТЕЙ
250   PRINT  HEX(03)  "ВВЕДИТЕ ПРАВЫЕ ЧАСТИ"
260   MAT  INPUT  B
270   REM  РЕШЕНИЕ СИСТЕМЫ
280   MAT  C=INV(C)
290   MAT  X=C*B
300   REM  ПЕЧАТЬ РЕШЕНИЯ
310   PRINT  HEX(03);"РЕШЕНИЕ"
320   MAT  PRINT X
```

При использовании этой программы могут возникнуть две проблемы. Во-первых, возможна потеря точности результата вследствие накопления ошибок округления при выполнении операций обращения и умножения матриц. Вторая проблема связана с неустойчивостью решения при малых изменениях элементов матрицы коэффициентов. Такие матрицы называются плохо обусловленными.

Первая ситуация наиболее опасна тогда, когда элементы главной диагонали имеют порядок, отличный от порядка остальных элементов, в частности если они близки к нулю. В этом случае рекомендуется поменять строки местами.

Для выявления ошибок, которые могут возникнуть при округлении, необходимо повторно обратить матрицу C в режиме непосредственного счета MAT C = INV(C), напечатать ее с помощью оператора MAT PRINT C и затем сравнить с исходной матрицей коэффициентов. Погрешность результата должна иметь порядок, равный средней разнице элементов исходной и дважды обращенной исходной матриц (при условии, что матрица C не является плохо обусловленной).

Для проверки матрицы на плохую обусловленность необходимо вычислить нормализованный определитель C1 матрицы C:

$$C1 = \frac{|C|}{A_1 A_2 \ldots A_n},$$

где

|C| —определитель матрицы коэффициентов C;
$A_k = (C^2_{k1} + C^2_{k2} + \ldots + C^2_{kn})$; $\kappa = 1, 2 \ldots\ldots n$;
n — размерность матрицы C.

Если окажется, что $C1 \ll 1$, то матрица C плохо обусловленная. Необходимо отметить, что величина определителя матрицы C не может служить признаком плохой обусловленности матрицы C.

Предыдущий пример программы решения системы линейных уравнений можно дополнить программой расчета нормализованного определителя. Для этого заменим строку 280 на строку 280 GOSUB 1000 и добавим следующий программный текст:

```
999  END
1000  REM  ПОДПРОГРАММА ВЫЧИСЛЕНИЯ НОРМАЛИЗОВАННОГО
1005  REM  ОПРЕДЕЛИТЕЛЯ
1010  A=1
1020  FOR  K=1  TO
1030  A1=0
1040  FOR  T = 1  TO
1050  A1=A1 + C(K,T)^2
1060  NEXT  T
1070  A=A*SQR(A1)
1080  NEXT  K
1085  REM  ОБРАЩЕНИЕ И ВЫЧИСЛЕНИЕ ОПРЕДЕЛИТЕЛЯ
1090  MAT  C=INV(C),  C1
1100  REM  ВЫЧИСЛЕНИЕ НОРМАЛИЗОВАННОГО ОПРЕДЕЛИТЕЛЯ
1110  PRINT  HEX(0A);  "НОРМАЛИЗОВАННЫЙ
ОПРЕДЕЛИТЕЛЬ=";  C1/A
1120  RETURN
```

Так, например, матрица

$$\begin{Vmatrix} 7 & 8 & 9 \\ 8 & 9 & 10 \\ 9 & 10 & 8 \end{Vmatrix}$$

является плохо обусловленной, так как ее нормализованный определитель равен 8.29 E-04.

Воспользовавшись приведенной выше программой с вектором правых частей, получим

$$\text{если } B = \begin{Vmatrix} 24 \\ 27 \\ 27 \end{Vmatrix}, \quad \text{то} \quad X = \begin{Vmatrix} 1{,}00 \\ 0{,}99 \\ 1{,}00 \end{Vmatrix}.$$

Если теперь незначительно изменить вектор правых частей, например

$$B = \begin{Vmatrix} 24{,}2 \\ 27{,}0 \\ 27{,}2 \end{Vmatrix}, \quad \text{то} \quad X = \begin{Vmatrix} -0{,}933 \\ 2{,}866 \\ 0{,}866 \end{Vmatrix}.$$

Малые изменения коэффициентов привели к значительному изменению результата. Заметим также, что в данном случае дважды обращенная матрица мало отличается от исходной

$$\begin{Vmatrix} 6{,}99 & 7{,}99 & 8{,}99 \\ 7{,}99 & 8{,}99 & 9{,}99 \\ 8{,}99 & 9{,}99 & 7{,}99 \end{Vmatrix}$$

т. е. результат не искажается из-за влияния ошибок округления при вычислениях,

12.2.10. ОПЕРАТОР MAT CON (ПРИРАВНИВАНИЕ К КОНСТАНТЕ)

Общая форма:

```
MAT  C=CON   [<а.в.1>[,<а.в.2>])]
```

где

C — имя числового массива;

<а.в.1>,<а.в.2> — выражения, определяющие новые размерности.

Оператор предназначен для приравнивания каждого элемента массива C к единице. Если новые размерности не указаны, они остаются такими же, как были указаны в операторах DIM или COM, по умолчанию они равны 10.

Пример 12.10

```
10  MAT  A=CON(2,2)
20  MAT  PRINT  A
```

В результате выполнения программы будет напечатано

```
1   1
1   1
```

12.2.11. ОПЕРАТОР MAT ZER (ОБНУЛЕНИЕ МАТРИЦЫ)

Общая форма:

```
MAT  C = ZER   [(<а.в.1>[,<а.в.2>])]
```

где

C — имя числового массива;
<а.в.1>,<а.в.2> — выражения, определяющие новую размерность массива C.

Оператор предназначен для приравнивания к нулю каждого элемента массива C.

Пример 12.11

```
10  MAT  C=ZER
20  MAT  B = ZER(5,6)
```

12.2.12. ОПЕРАТОР MAT IDN (СОЗДАНИЕ ЕДИНИЧНОЙ МАТРИЦЫ)

Общая форма:

```
MAT  C=IDN   [(<а.в.1>,<а.в.2>)]
```

где

C — имя числового массива;
<а.в.1>,<а.в.2> — выражения, определяющие новые размерности.

Оператор предназначен для превращения квадратной матрицы C в единичную. Если матрица C не является квадратной, то выполнение программы прекращается с сообщением об ошибке.

Пример 12.12.

```
10  MAT  A=IDN(3,3)
20  MAT  PRINT  A;
```

В результате выполнения этой программы будет напечатано

```
1   0   0
0   1   0
0   0   1
```

12.2.13. ОПЕРАТОР MAT TRN (ТРАНСПОНИРОВАНИЕ МАТРИЦЫ)

Общая форма:

```
MAT  C=TRN(A)
```

где

C и A — имена числовых матриц.

Оператор предназначен для транспонирования матриц. Результат помещается в матрицу C. Размерность матрицы C определяется по размерности матрицы A. Имена матриц по обе стороны знака равенства должны быть различными.

Пример 12.13

```
10   DIM A(3,2)
20   MAT READ A : MAT PRINT A;
30   MAT C = TRN (A)
40   MAT PRINT C;
50   DATA 9,8,7,5,4,3,2,1
```

В результате выполнения программы будет напечатано:

а) содержимое исходной матрицы A

```
9   8   7
6   5   4
3   2   1
```

б) содержимое транспонированной матрицы C

```
9   6   3
8   5   2
7   4   1
```

12.2.14. ОПЕРАТОР MAT REDIM (ПЕРЕОПРЕДЕЛЕНИЕ РАЗМЕРНОСТИ МАССИВА)

Общая форма:

MAT REDIM{< имя числового массива > (< а.в.1 > [,< а.в.2 >])
< имя символьного массива > (< а.в.1 > [,< а.в.2 >])}[< а.в.3 >]

где

<а.в.1>,<а.в.2> — выражения, определяющие новые размерности;
<а.в.3> — выражение, определяющее максимальную длину элемента символьного массива.

Оператор позволяет изменять размерности массивов. Однако одномерный массив не может быть переопределен как двумерный, и наоборот. При этом переопределенный массив не должен занимать больше места, чем было первоначально для него отведено в операторе DIM или COM.

Пример 12.14

```
10  MAT  REDIM  A(4,5)
20  MAT  REDIM  A1(X+3,Y1)
30  MAT  REDIM  B⊙(5,4)25
40  MAT  REDIM  C⊙ (5,K*4)X
```

Глава 13. РАБОТА С СИМВОЛЬНЫМИ ДАННЫМИ

13.1. Шестнадцатеричная функция

В микроЭВМ «Искра 226», как и в большинстве ЭВМ, биты объединяются в группы по 8, что соответствует 1 байту информации. Возможны $2^8 = 256$ различные комбинации состояний битов в байте. Если обозначать состояние бита 0 или 1, то каждому состоянию бита будет соответствовать двоичное число, т. е. число, записанное с использованием цифр 0 и 1. Например, 00001000, 00011010 и т. д.

Для записи двоичных чисел, так же как и привычных десятичных чисел, пользуются позиционной системой счисления. Эта система основана на принципе позиционного, или поместного, значения цифр: значение каждой цифры зависит от ее места в числе.

Число в позиционной системе счисления записывается следующим образом:

$A_n A_{n-1} \ldots A_1 A_0$,

Это означает, что число равно

$A_n P^n + A_{n-1} P^{n-1} \ldots + A_1 P^1 + A_0 P^0$,

где

P — основание системы счисления; A_i — цифра в разряде i.

Используемые цифры должны быть меньше основания системы:

$0 \le A_i \le P-1$.

Например, двоичное число

00111101

представляет десятичное число

$0 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 61$.

Таким образом, каждая комбинация битов в байте представляется двоичным числом от 0 до 255. Однако использовать двоичные числа для записи состояний байта неудобно из-за большой их длины даже для относительно небольших чисел. С другой стороны, привычные десятичные числа не позволяют отразить состояние байта. Дело в том, что между двоичными и десятичными разрядами нет взаимно однозначного соответствия. Скажем, для представления четырех двоичных разрядов (числа от 0 до 15) достаточно двух десятичных. Но в двух десятичных разрядах в свою очередь могут быть представлены числа от 0 до 99, требующие 7 двоичных разрядов.

Для записи состояния байтов в микроЭВМ «Искра 226» используется шестнадцатеричная система. Один разряд числа в шестнадцатеричной системе в точности соответствует четырем двоичным разрядам. В шестнадцатеричной системе используются следующие 16 цифр

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

Ниже приведено соответствие между двоичными, десятичными и шестнадцатеричными числами.

Двоичное число	Десятичное число	Шестнадцатеричное число
0000	000	00
0001	001	01
0010	002	02

0011	003	03
0100	004	04
0101	005	05
0110	006	06
0111	007	07
1000	008	08
1001	009	09
1010	010	0A
1011	011	0B
1100	012	0C
1101	013	0D
1110	014	0E
1111	015	0F

Любое состояние байта может быть представлено двухразрядным шестнадцатеричным числом от 00 до FF, причем младшая шестнадцатеричная цифра (16^0) в точности соответствует четырем младшим битам (от 2^0 до 2^3), а старшая (16^1)—четырем старшим битам (2^4—2^7). Например, рассмотренное ранее двоичное число

00111101

в шестнадцатеричной системе представляется как

3D

что соответствует

$3 \cdot 16^1 + D \cdot 16^0 = 3 \cdot 16^1 + 13 \cdot 16^0 = 61.$

Итак, вся информация хранится в машине в виде двоичных чисел. Для того чтобы обрабатывать на ЭВМ, скажем, символьные данные, необходимо поставить в соответствие каждому символу определенное число. При вводе символа, например путем нажатия клавиши с обозначением этого символа, в ЭВМ посылается соответствующий ему числовой код. При выводе кода он в свою очередь преобразуется в соответствующий символ на экране или печатающем устройстве.

Для кодирования символьных данных в микроЭВМ «Искра 226» используется стандартный код КОИ-8 (Код Обмена Информацией— 8-разрядный). Кодовая таблица, задающая соответствие между символами и их кодами, приведена в приложении 11 Каждый столбец таблицы соответствует определенному значению четырех старших разрядов, заданному в двоичной и десятичной форме; каждая строка — определенному значению четырех младших разрядов. Код любого символа, приведенного в таблице, складывается из номера столбца, который дает старшие разряды кода и номера строки, представляющего младшие разряды кода. Наоборот, для определения символа по коду необходимо отыскать столбец, соответствующий значению старших разрядов, и строку, соответствующую значению младших разрядов. На их пересечении и будет находиться искомый символ. Например, коду ООП 1101 (3D) соответствует символ =, находящийся на пересечении 3-го столбца и 13-й строки.

Из кодовой таблицы КОИ-8 видно, что не всем кодовым комбинациям соответствуют алфавитно-цифровые символы или специальные знаки. Ряд кодов, в частности от 00 до 20, используется в качестве кодов команд управления внешними устройствами, которые обозначены в таблице мнемокодами. Так, например, ПС (код 0А) — Перевод Строки. ВК (код 0D) — Возврат Каретки и т. д. Следует отметить, что одни и те же управляющие коды могут иметь различный смысл для различных внешних устройств. Дисковые устройства не распознают управляющих кодов в символьных данных, которые записываются так же, как и все остальные. При выводе символьных данных на экран и печатающие устройства стирание с экрана, перевод бумаги и т. д. осуществляются по определенным управляющим кодам.

В Бейсике микроЭВМ «Искра 226» пользователю предоставляется возможность задавать символьные константы не только как строки символов, но и непосредственно в виде последовательностей шестнадцатеричных кодов. В этом случае используется шестнадцатеричная функция HEX.

HEX (< последовательность шестнадцатеричных кодов >)

Значением этой функции является строка символов, соответствующих этим кодам. Например, символ = может быть задан непосредственно или своим кодом в виде

HEX(61)

Аналогичным образом вместо слова «БЕЙСИК,» можно задать функцию HEX, перечислив в скобках коды символов «Б», «Е», «Й», «С», «И», «К».

HEX(E2E5EAF3E9EB)

Отсюда вытекает, что вместо

```
A⊙ = "БЕЙСИК"
```
можно писать
```
A⊙ = HEX  (E2E5EAF3E9EB)
```
или вместо
```
PRINT  "БЕЙСИК"
```
писать
```
PRINT  HEX(E2E5EAF3E9EB)
```

Шестнадцатеричной функцией можно пользоваться во всех случаях, где допустимо использование символьных констант: в операторах LET, PRINT, DATA, при вводе с клавиатуры по оператору INPUT и т. д. Данные, заданные функцией HEX, можно комбинировать с данными в виде строк символов в кавычках, например

```
PRINT  "БЕЙ";  HEX(F3E9EB)
```

Ниже приведены несколько примеров использования функции HEX в различных операторах:

```
A⊙ = HEX(030A0A)
DATA  "СЛОВО",  HEX(11),  HEX(12)
IF D⊙  HEX(08)  THEN  210
GOSUB'  50  (4,  "ИМЯ",  HEX (19))
```

При задании значений символьных переменных с помощью функции HEX, так же как и в случае обычных символьных констант, присваиваемые значения заносятся, начиная с крайнеголевого символа (байта) символьной переменной. Если присваиваемые значения короче, то остальные символы считаются пробелами.

Использование функции HEX позволяет пользователю задать с клавиатуры любой код, даже если на клавиатуре нет клавиши с символом, имеющим данный код, или если клавиша с данным кодом воспринимается как команда управления.

Например, применение функции HEX позволяет ввести в текст оператора код клавиши CR/LF HEX(85)

```
DEFFN'30  "LIST  100,200";  HEX(85)
```

В результате нажатия клавиши специальных функций 30 обеспечивается выполнение тех же функций, что и при нажатии клавиш

```
LIST  100,200
CR/LF
```

Часто функцию HEX используют для задания кодов управления экраном, которые также не могут быть введены с клавиатуры. Например, как отмечалось выше, вывод кода HEX (03)

```
PRINT  HEX(03)
```

очищает экран и переводит курсор в левый верхний угол.

Иногда шестнадцатеричную функцию приходится использовать и для задания обычного текста. Например, необходимо выдать на экран текст, содержащий кавычки

```
АЭРОПОРТ "ВНУКОВО"
```

Записать кавычки в символьной константе программы нельзя, так как они служат для задания самой константы. Кавычки в этом случае можно задать с помощью шестнадцатеричного кода кавычек HEX(22)

```
PRINT  "АЭРОПОРТ";HEX (22) ;"ВНУКОВО";HEX (22)
```

13.2. Строки и подстроки символов

Как известно, значением символьной переменной является последовательность символов, или, в общем случае, последовательных любых шестнадцатеричных кодов. Длина этой строки в символах (или, что то же самое, в байтах) задается в операторе DIM или COM, а по умолчанию полагается равной 16. В ряде операторов, однако, возникает необходимость оперировать какой-либо частью значения символьной переменной, т. е. ее подстрокой. Для этого в Бейсике имеется специальная функция выделения подстроки STR.

Например, функция

```
STR(A⊙,6,3)
```

имеет своим значением подстроку значения переменной А⊙, начинающую с шестого символа значения А⊙ и имеющую длину три символа, т. е. шестой, седьмой и восьмой символы значения переменной А⊙. Если, скажем, значением переменной А⊙ была строка «МИНИ-ЭВМ»

```
:А⊙ = "МИНИ-ЭВМ"
:PRINT  А⊙
МИНИ-ЭВМ
:__
```

то значением STR(А⊙,6,3) будет «ЭВМ»

```
: PRINT  STR (А⊙ ,6,3)
ЭВМ
:__
```

Функция STR(В⊙,10) задает подстроку значения переменной В⊙, начиная с десятого байта, и по умолчанию (длина подстроки в функции не задана) до последнего шестнадцатого байта переменной В⊙, т. е. длиною семь байт

```
:В⊙ = "МАТЕРИАЛ № 50"
: PRINT В⊙
МАТЕРИАЛ № 50
:PRINT   STR(В⊙,10)
№ 50
:__
```

Функция STR может играть роль символьных переменных в операторах Бейсика. Например, функция STR может стоять слева от знака = в операторе LET, т. е. с ее помощью можно присваивать значение части символьной переменной

```
:10  К⊙ = "МАТЕРИАЛ:"
:20  М⊙ = "КИРПИЧ"
:30  STR(К⊙,11) = М⊙
:40  PRINTК⊙
:RUN
МАТЕРИАЛ:   КИРПИЧ
:__
```

Общая форма функции STR:

```
STR(<символьная переменная>,   <а.в.1>   [,<а.в.2>])
```

В качестве символьной переменной может использоваться простая символьная переменная, элемент символьного массива или обозначение символьного массива.

Целая часть первого арифметического выражения используется в качестве начального номера байта выделяемой подстроки. Целая часть второго арифметического выражения, которое не является обязательным, определяет количество байтов в выделяемой подстроке, если второе выражение отсутствует. Естественно, значение выражения 1 должно быть в пределах от единицы до числа байт символьной переменной, а значение выражения 2 — от единицы до числа оставшихся байт символьной переменной.

При использовании обозначения символьного массива в качестве аргумента функции STR, например

```
STR(А⊙ ( ),12,85)
```

символьный массив рассматривается как одна непрерывная строка, образованная последовательным соединением всех его элементов (по строкам). Длина этой строки, следовательно, равна произведению числа элементов массива на длину элемента. Таким образом, с помощью функции STR можно задавать символьные строки, длина которых больше 253 байт (максимальная длина простой символьной переменной или элемента символьного массива).

Пусть, например, определен символьный массив, состоящий из 10 строк и 2 столбцов, с длиною элемента, равной 50 символам

```
10  DIM С⊙ (10,2)50
```

тогда можно выполнять операции со строкой, равной суммарной длине элементов массива 10×2×50=1000 байт, записав

```
STR(C⊙ ( ),1)
```

или с любой меньшей строкой

```
STR (C⊙ ( ),1,810)
STR (C⊙ ( ) ,200,641)
```

Пример 13.1

Следующая программа присваивает значения элементам массива

```
:10  REM  ПРИСВАИВАНИЕ ЗНАЧЕНИЙ ЭЛЕМЕНТАМ
:20  DIM  B⊙(2,3)2
:30  B⊙(1,1)="АБ"
:40  B⊙(1,2)="ВГ"
:50  B⊙(1,3)="ДЕ"
:60  B⊙(2,1)-"ЖЗ"
:70  B⊙(2,2)="ИК"
:80  B⊙(2,3)="ЛМ"
:RUN
```

При использовании функции STR значения элементов массива рассматриваются как одна непрерывная строка, из которой можно выделять любые подстроки

```
:PRINT  STR(B⊙( ),1)
АБВГДЕЖЗИКЛМ
:PRINT  STR(B⊙( ),4,8)
ГДЕЖЗИКЛ
:PRINT  STR(B⊙( ),6,3)
ЕЖЗ
:—
```

Объединение элементов массива в одну строку с помощью функции STR дает возможность выполнять операции сразу со всеми или несколькими последовательными элементами массива.

Пример 13,2

Пусть требуется присвоить каждому элементу массивов A⊙() и B⊙ () значение соответствующего элемента массива C⊙(). Ниже приведен фрагмент программы, обеспечивающий присваивание значений элементам с использованием операторов цикла

```
10  REM ПРИСВАИВАНИЕ  ЗНАЧЕНИЙ МАССИВАМ
20  DIM A⊙(10),B⊙(10),C⊙(10)
.  .  .
100  REM ПРИСВАИВАНИЕ С  ПОМОЩЬЮ ОПЕРАТОРОВ  ЦИКЛА
110  FOR  I=1  TO  10
120  A⊙(I),B⊙(I)=C⊙(I)
130  NEXT  I
```

Вместо поэлементного присваивания значений в цикле (строки 110—130) можно осуществить присваивание значений сразу всем элементам с помощью одного оператора

```
100  REM ПРИСВАИВАНИЕ С ПОМОЩЬЮ ФУНКЦИИ STR
110  STR(A⊙( ),1),STR(B⊙( ),1)=STR(C⊙.( ),1)
```

Так как длины элементов всех массивов одинаковы, при присваивании значения одного массива другому массиву границы соответствующих элементов совпадают.

Очевидно, используя функцию STR можно осуществить присваивание и части элементов, например четырем элементам — со второго по пятый

```
110  STR(A⊙( ),11,64),STR(B⊙( ),11,64) =STR(C⊙( ),11)
```

Следует иметь в виду, что операция присваивания значения одной переменной другой выполняется системой по шагам, байт за байтом, начиная с левого крайнего. Поэтому если слева и справа от знака равенства в оператора LET указаны различные подстроки одной и той же переменной, то значение подстроки справа может изменяться в процессе выполнения операции присваивания.

Пример 13.3

```
:10  REM ИЛЛЮСТРАЦИЯ ПОБАЙТНОГО ПРИСВАИВАНИЯ.
:20  DIM A⊙(2,2)2
:20  STR(A⊙( ),1,1)="X"
```

```
:30  PRINT STR(A⊙( ),l)
:40  STR(A⊙( ),2)=STR(A⊙( ),l)
:50  PRINT STR(A⊙( ),l)
:RUN
x
xxxxxxxx
:__
```

13.3. Присваивание начальных значений символьным переменным

При выполнении оператора запуска программы RUN каждая символьная переменная получает начальное значение, равное пробелам, т. е. все байты этих переменных заполняются кодами HEX (20). Иногда, однако, возникает необходимость занести во все байты символьной переменной или некоторой ее подстроки один и тот же символ или, в общем случае, код, отличный от пробела.

В этом случае можно, конечно, непосредственно повторить необходимое число раз соответствующий символ или код при задании константы

```
10  A⊙ = HEX (08080808080808080808080808080808)
```

или воспользоваться оператором цикла

```
10  FOR  I=1  TO  16
20  STR(A⊙,I,1)=HEX(08)
30  NEXT  I
```

В обоих примерах во все шестнадцать байт переменной A⊙ заносится код HEX(08). Еще более громоздко выглядит это в случае массива

```
10  DIM  K⊙(10,15)3
20  FOR  I=1  TO 10
30  FOR  J=l  TO 15
30  K⊙(I,J) = "000"
40  NEXT  J
40  NEXT  I
```

Здесь можно, правда, воспользоваться приемом, использованным в примере 13.3.

```
10  DIM  K⊙(10,15)3
20  STS(K⊙( ),1,1)="0"
30  STR(K⊙( ),2) = STR(K⊙( ),1)
```

Для задания одинаковых значений во все байты символьных переменных или их подстрок в Бейсике имеется специальный оператор INIT. После слова INIT в скобках задается значение, а после скобок перечисляются через запятую переменные, в которые должно быть занесено это значение.

Для задания значения в виде кода в скобках нужно указать две шестнадцатеричные цифры. Например, оператор

```
INIT(0A)  R⊙
```

заносит во все байты переменной A⊙ код HEX (08). В скобках может быть указан символ в кавычках

```
INIT("1")  Q⊙(1),F⊙
```

или символьная переменная

```
INIT(M⊙) N⊙,X⊙,19⊙
```

В последнем случае для задания значения используется первый байт символьной переменной.

В списке переменных оператора INIT можно задавать обозначения символьных массивов без использования функции STR

```
INIT(".")  C⊙( ),  G1⊙( ),F⊙
```

При этом значение присваивается всем байтам символьного массива.

При использовании функции STR значение может быть занесено в определенные байты переменных или массивов, например

```
INIT("/")  STR(S⊙( ),1,20),STR(S⊙( ),22),STR(D⊙,5)
```

13.4. Длина строки символов

До сих пор под длиной символьной переменной подразумевалось строго определенное число байтов, зарезервированных системой для хранения значения этой переменной. Часто, однако, переменным присваиваются более короткие значения. Эти значения заносятся, начиная с крайнего левого байта, а оставшиеся незаполненными крайние справа байты автоматически заполняются кодами пробела. Для определения числа символов в значении символьной переменной без учета концевых пробелов в Бейсике имеется функция LEN.

Аргументом функции LEN является строка символов, заданная в виде константы, переменной или обозначения массива; значением функции — число, равное количеству символов в значении символьной переменной, начиная с крайнего левого байта и до последнего не равного пробелу байта включительно. Если значением символьной переменной является строка из всех пробелов, то значением функции LEN будет 1, т. е. все пробелы, кроме первого, рассматриваются как концевые. Примеры задания функции:

```
LEN(A⊙)
LEN(B⊙(3))
LEN(STR(G⊙( ),2,8)
LEN("ABC")
LEN(M⊙( ))
```

Функция LEN как числовая функция может использоваться везде, где допускается использование числовых данных, например

```
IF  LEN (A⊙) =5   THEN   120
H%=LEN(A⊙)/\2
```

Рассмотрим некоторые примеры использования функции LEN.

Значения символьных переменных выводятся на печать вместе с концевыми пробелами. Это препятствует выдаче слитного текста из значений отдельных символьных переменных или элементов массива. Функция LEN в сочетании с функцией STR позволяет обеспечить вывод слитного текста независимо от наличия концевых пробелов.

Пример 13.4

Введем и выполним следующую программу ввода значений двух символьных переменных и печати их в плотном формате:

```
:10   REM ПРОГРАММА ПЕЧАТИ СЛОВ ЧЕРЕЗ ПРОБЕЛ
:20   INPUT "ПЕРВОЕ СЛОВО",A⊙
:30   INPUT "ВТОРОЕ СЛОВО",B⊙
:40   PRINT A⊙; B⊙
RUN
ПЕРВОЕ СЛОВО?МИНЗДРАВ
ВТОРОЕ СЛОВО?СССР
МИНЗДРАВ      СССР
:__
```

Воспользуемся функциями LEN и STR для печати слов с одним пробелом между ними. Заменим строку 40 на следующую:

```
:40   PRINT  STR.(A⊙,1,LEN,(A⊙) + 1);B⊙
:RUN
ПЕРВОЕ СЛОВО?МИНЗДРАВ
ВТОРОЕ СЛОВО?СССР
МИНЗДРАВ  СССР
:__
```

Функция LEN позволяет контролировать длину строк, вводимых пользователем.

Пример 13.5

Следующая программа использует для ввода значений помеченную подпрограмму ´98. При обращении в подпрограмму передается текст подсказывающего сообщения и требуемая длина вводимой строки.

```
10   REM ПРОГРАММА С ПОДПРОГРАММОЙ ВВОДА'
20   DIM T⊙30
30   GOSUB'("ИМЯ ФАЙЛА",8)
```

```
40  GOSUB'("КОД ПРОДУКЦИИ",8)
50  GOSUB'("ПРОДОЛЖИТЬ (ДА,НЕТ)",8 )
. . .
1000  REM ПОДПРОГРАММА ВВОДА
1010  DEFFN'98(T⊙,L)
1020  PRINT STR(T⊙,1,LEN(T⊙));
1030  INPUT A⊙:REM ВВОД СТРОКИ
1040  IF LEN(A⊙)=L THEN 1070
1050  PRINT "НЕВЕРНАЯ ДЛИНА СТРОКИ"´
1060  GOTO 1020
1070  RETURN
```

В строках 30, 40 и 50 записаны обращения к подпрограмме ввода. Сама подпрограмма начинается со строки 1010. В качестве формальных параметров в ней используются символьные переменные T⊙ (подсказывающее сообщение) и L (требуемая длина символов при вводе).

Подпрограмма сначала выдает значение переменной T⊙, т. е. подсказывающее сообщение, без концевых пробелов и без перевода курсора (строка 1020). Затем по оператору INPUT выводится знак «?» и вводится набранное на клавиатуре значение. По оператору условного перехода осуществляется возврат из подпрограммы, если длина введенной строки равна значению переменной L, т. е. требуемой длине. В противном случае выводится сообщение о неверной длине введенной строки и снова повторяется выполнение операторов со строки 1020.

В следующем примере функция LEN используется для выравнивания выводимого текста по правому краю.

П р и м е р 13.6 Введем в оперативную память следующую программу:

```
:10  REM ПРОГРАММА ВЫРАВНИВАНИЯ ПО ПРАВОМУ КРАЮ
:20  FOR  I=1  TO  3
:30  READ  A⊙
:40  PRINT  TAB(40-LEN(A⊙));STR(A⊙,1)
:50  NEXT  I
:60  DATA "НАИМЕНОВАНИЕ","МАТЕРИАЛ","КОД"
```

При выполнении этой программы будет трижды в цикле считываться и печататься очередное значение, заданное в операторе DATA. Всякий раз при этом функция TAB будет устанавливать начальную позицию печати на столько позиций левее позиции 40, сколько занимает слово. В результате концы всех слов будут выравнены по позиции 40.

```
RUN
                              НАИМЕНОВАНИЕ
                                  МАТЕРИАЛ
:_                                    КОД
```

Функция LEN может также использоваться для определения длины символьной константы, например

```
:PRINT   LEN(HEX(202122818283))
6
:_
```

13.5. Вывод шестнадцатеричных символьных кодов

В ряде случаев, например для контроля при отладке программ, необходимо знать содержимое символьных переменных. Казалось бы, можно воспользоваться оператором PRINT и выводить непосредственно строки символов, являющиеся значениями интересующих символьных переменных.

Однако, во-первых, некоторым различным кодам на данном устройстве вывода могут соответствовать близкие или даже одинаковые по начертанию символы (например, некоторые буквы Русского и латинского алфавитов).

Во-вторых, некоторым кодам не соответствуют никакие символы на устройстве вывода и они выводятся как пробелы.

В-третьих, некоторые коды воспринимаются устройствами вывода как команды управления (перемещение курсора или бумаги, стирание с экрана, выдача звукового сигнала и т. п.).

Таким образом, с помощью оператора PRINT удастся распознать в лучшем случае только часть

кодов, содержащихся в символьной переменной. Если, например, в одном из байтов переменной содержится код очистки экрана HEX(03), то после вывода символов, коды которых записаны в предыдущих бантах, эти символы будут стерты с экрана. Поскольку к тому же вывод значения переменных осуществляется достаточно быстро, то вообще не удастся увидеть ни одного символа переменной, выданного до кода HEX (03).

В Бейсике «Искры 226» предусмотрен специальный оператор вывода шестнадцатеричных кодов значений символьных переменных HEXPRINT. При выполнении этого оператора распечатываются коды содержимого каждого байта символьной переменной или каждого символа символьной константы в виде пар шестнадцатеричных цифр, например

```
:HEXPRINT   "ABC"
414243
:A⊙ = "ABC"
:HEXPRINT  A⊙
41424320202020202020202020202020
:INIT(03)   A⊙
:HEXPRINT  A⊙
03030303030303030303030303030303
```

или

```
:10  DIM A⊙10
:20  STR(A⊙,2)="ПРАКТИКУМ"
:30 STR(A⊙,1,1)=HEX(03)
:40 HEXPRINT A⊙
RUN
03F0F22E1EBF4E9EBF5ED
```

В операторе HEXPRINT может быть указано обозначение символьного массива, например

```
HEXPRINT B⊙ ( )
```

В этом случае выводятся шестнадцатеричные коды сразу всех элементов массива.

В операторе HEXPRINT могут перечисляться несколько элементов

```
HEXPRINT  K⊙;E⊙(5),   STR(A⊙( ),1,10),B⊙( )
```

При использовании в качестве разделителя точки с запятой шестнадцатеричные коды распечатываются вплотную без пробелов между переменными. Если в качестве разделителя используется запятая, то шестнадцатеричные коды следующего элемента печатаются с новой строки. В операторе HEXPRINT, так же как и в операторе PRINT, можно использовать функцию TAB для печати с определенной позиции строки.

Длина последовательности кодов, выводимых по оператору HEXPRINT, определяется числом байтов в соответствующей символьной константе или переменной. Поскольку значение каждого байта выводится двумя шестнадцатеричными цифрами, то длина выводимой строки, естественно, будет вдвое больше длины константы или переменной.

13.6. Преобразование числовых данных в символьное представление

В Бейсике микроЭВМ «Искра 226» выделены два типа данных— числовые и символьные. Эти типы различаются формой представления данных в памяти машины и выполняемыми над ними операциями.

Для представления чисел принята специальная форма внутреннего представления, которая обеспечивает компактное хранение и облегчает выполнение арифметических действий. С другой стороны, последовательность знаков числа, очевидно, может рассматриваться как обычная последовательность символов и храниться в символьной форме. В этом случае можно выполнять различные операции с символами числа, но исключается выполнение арифметических действий.

В ряде случаев, однако, возникает необходимость осуществлять и символьную обработку числа, и выполнять с ним арифметические действия. Это требует наличия в языке средств преобразования чисел из символьной формы в числовую и обратно.

Непосредственно использовать оператор присваивания для присваивания значений числовых переменных символьным и символьных числовым, как уже говорилось, нельзя. Для этой цели в Бейсике имеется специальный оператор CONVERT. С его помощью число, хранимое как последовательность символов, может быть преобразовано в числовой формат, т. е. в значение числовой переменной. И

наоборот, число, хранимое в числовом формате, может быть преобразовано в последовательность составляющих его символов, т. е. в значение символьной переменной.
Введем и выполним следующую программу:

```
:10  A⊙="1200.50"
:20  CONVERT A⊙ TO A
:30  PRINT A⊙;A;2*A
:RUN
1200.50              1200.5       2400
```

В строке 10 символьной переменной A⊙ присваивается значение—строка символов «1200.50», т. е. последовательность из цифр и точки. Оператор CONVERT в строке 20 преобразует значение символьной переменной A⊙, указанной после слова CONVERT, в значение числовой переменной A, указанной после слова TO (результат операции). По оператору PRINT в строке 30 печатается значение символьной переменной A⊙ с выводом концевых пробелов, затем числовой переменной A (незначащие нули не выводятся) с пробелом вместо знака числа и пробелом после числа, а также значение числовой переменной, умноженное на 2 с пробелом вместо знака.

Естественно, что при преобразовании числа из символьной формы в числовую преобразуемое символьное значение переменной должно представлять собой правильную, с точки зрения Бейсика, запись числа. Число должно быть записано или в естественной, или в экспоненциальной форме, оно может содержать знак, не более 13 десятичных цифр, десятичную точку, символ E для отделения мантиссы от показателя, знак и две цифры для обозначения показателя степени. В противном случае выполнение оператора CONVERT прервется из-за ошибки.

Допустим, символьной переменной A⊙ присвоено следующее значение:

```
:A⊙ = "-123.05E+14"
```

Преобразуем эту строку в числовую форму

```
:CONVERT  A⊙   TO  X
:PRINT  X
-1.23050000E+16
:__
```

Однако первые три символа значения A⊙ также представляют правильную запись числа, и, следовательно, могут быть преобразованы в числовую форму

```
:CONVERT  STR(A⊙,1,3)   TO  X
: PRINT X
–123
:__
```

Правильной записью числа являются и 10 первых символов, т, е, до символа 4:

```
:CONVERT   STR.(A⊙,1,10)   TO  X
: PRINT X
– 1.23050000E + 16
:__
```

Строка, содержащая последовательность символов числа, может, кроме того, содержать различные другие символы. В этом случае, прежде чем выполнять преобразование, необходимо выделить последовательность символов, представляющую правильную запись числа. Для облегчения решения этой задачи в Бейсике имеется функция NUM.

Аргументом функции NUM является символьная строка в форме константы, переменной или обозначения массива, а значением—число, равное длине строки, начинающейся с первого байта и содержащей правильную запись числа

```
NUM(A⊙)
NUM(STR(B⊙( ),T,20)
NUM("12.2+14.8+18.5")
NUM(Q⊙ ( ))
```

При вычислении значения функции последовательно просматриваются байты аргумента до тех пор, пока их содержимое соответствует допустимой последовательности символов в записи числа. Число

таких байт и является значением функции. Пробелы, предшествующие правильной записи числа и следующие после нее, включаются в общее количество. Если строка не содержит правильной записи числа, начинающейся с левого края, то значением функции будет 0. Рассмотрим пример:

```
:10  REM ПРОГРАММА ИЛЛЮСТРАЦИИ ФУНКЦИИ NUM
:20  INPUT  A⊙
:30  PRINT  NUM(A⊙
:RUN
+ 1.2   -14   +1.2587
5
:RUN
98.1E+10
16
:RUN
0..
2
```

Пример 13.7

В следующей программе число вводится в символьной форме, а функция NUM используется для контроля правильности ввода перед последующим преобразованием в числовую форму.

```
:10  REM ПРОГРАММА СО ВВОДОМ ЧИСЛА В СИМВОЛЬНОЙ ФОРМЕ
:20  INPUT  "ЧИСЛО",A⊙
:30  IF NUM(A⊙)=16  THEN  60
:40  PRINT  "ОШИБКА В ЗАПИСИ ЧИСЛА"
:50  GOTO  20
:60  CONVERT  A⊙  TO  F
:70  PRINT  "число=;F
```

По умолчанию длина символьной переменной A⊙ равна 16 байтам. Если введена корректная последовательность символов числа, то значение функции NUM должно равняться 16, поскольку Учитываются конечные пробелы. В этом случае осуществляются преобразование в числовую форму и вывод числового значения.

Если же A⊙ содержит хотя бы один некорректный символ, значение функции NUM. будет меньше 16. В этом случае выдается сообщение об ошибке.

```
:RUN
ЧИСЛО?123 +
ОШИБКА  В  ЗАПИСИ  ЧИСЛА
ЧИСЛО?+123
ЧИСЛО = 123
:__
```

Наличие средств преобразования чисел из символьной формы в числовую позволяет осуществлять ввод любых данных в символьной форме. Это дает возможность унифицировать ввод и контроль вводимых значений с использованием средств обработки символьных данных.

Пример 13.8

```
10  REM  ПРОГРАММА С ПОДПРОГРАММОЙ ВВОДА ЧИСЕЛ И
ТЕКСТОВ
20  DIM  P⊙30.R⊙30
. . .
50  REM  ВВОД ЧИСЛА
60  G0SUB'80("ИНВЕНТАРНЫЙ НОМЕР",1,1,9999,2)
70  REM  ВВОД ТЕКСТА
80  GOSUB'80("НАИМЕНОВАНИЕ КНИГИ,3-16 ЗН.", 2, 3,16, 0)
90  REM  ВВОД ЧИСЛА
100  G0SIJB'80("СТОИМОСТЬ",l,l,999,2)
. . .
1000  REM  ПОДПРОГРАММА ВВОДА
1010  DEFFN'80(P⊙,T,M1,M2,M3)
1020  PRINT  STR(P⊙, l, LEN(P⊙)+l);
1030  INIT(" ")R⊙
1040  INPUT STR(R⊙,2)
```

```
1050  ON T GOTO 1070,1160
1060  REM  КОНТРОЛЬ И ПРЕОБРАЗОВАНИЕ ЧИСЕЛ
1070  IF  NUM(R⊙)<>30 THEN 1130:REM НЕКОРРЕКТНОЕ ЧИСЛО
1080  CONVERT R⊙ TO R
1090  IF  ABS(2*R-M2-M1)>M2-M1 THEN 1130:REM ВНЕ ДИАПАЗОНА
1100  IF  INT(R*10^M3)<>R*10^M3 THEN 1130:REM ВНЕ ФОРМАТА
1110  RETURN
1120  REM  ОБРАБОТКА ОШИБКИ
1130  PRINT  "НЕВЕРНО, ПОВТОРИТЕ!"
1140  GOTO 1020
1150  REM  КОНТРОЛЬ ТЕКСТА
1160  IF ABS(2*(LEN(R⊙)-1)-M2-M1)>M2-M1 THEN
1130:REM НЕВЕРНАЯ ДЛИНА
1170  R⊙=STR(R⊙,2)
1180  RETURN
```

Подпрограмма ввода начинается с программной строки 1010, обращения к ней имеются в строках 60, 80 и 100. Подпрограмма использует следующие пять параметров:

переменная P⊙ содержит текст подсказывающего сообщения при вводе;

переменная T содержит 1, если должно вводиться число, 2, если должен вводиться текст;

переменная Ml задает минимальное допустимое значение вводимого числа или минимальную длину вводимого текста;

переменная M2 задает максимальное допустимое значение вводимого числа или максимальную длину вводимого текста;

переменная M3 задает допустимое количество дробных десятичных разрядов числа, при вводе текстов не используется.

Собственно ввод осуществляется в строке 1040 в переменную R⊙, причем начиная со второго байта. Это сделано для того, чтобы в последующем можно было использовать функцию LEN для проверки соответствия длины введенного текста, начиная с 1, так как в случае заполненной одними пробелами строки функция LEN принимает значение 1. Поэтому и при отсутствии ввода, и при вводе в первый байт одного символа ее значение будет равно 1.

По оператору ON-GOTO в строке 1050 осуществляется переход к числовому или символьному контролю. При невыполнении условий проверки выполняются операторы в строках ИЗО и 1140 — выдается сообщение об ошибке и осуществляется переход к повторному вводу.

Для выполнения обратного преобразования, т. е. преобразования значения числовой переменной или арифметического выражения в значение символьной переменной также используется оператор CONVERT, в котором необходимо дополнительно указать формат представления числа в символьной форме.

При преобразовании числа из символьной формы в числовую не возникает вопроса о формате представления числа после преобразования, поскольку система выбирает его автоматически в зависимости от размерности самого числа. В случае же обратного преобразования возможны различные символьные представления одного и того же числа.

Например, число, имеющее в числовой форме стандартный вид 3207.45, в символьной форме может быть представлено как

```
3207.45
000003207.45
3207.45000
3.207E + 03
```

и т. д.

Общая форма оператора CONVERT при преобразовании из числовой в символьную форму.

CONVERT <а.в.> TO <символьная переменная>, (<формат>)

где

<формат> = [$\begin{Bmatrix}+\\ \\ -\end{Bmatrix}$] [# #...#] [.] [# #...#] [/\/\/\]

Формат определяет представление числа в символьной форме. В случае отсутствия знака в формате число записывается без знака, при указании знака «плюс» числу предшествует + или —, а при указании

знака «минус» числу предшествует пробел или минус.

Символы # определяют число цифр, а символы «.»— положение десятичной точки.

Признак /\/\/\ задает запись числа в экспоненциальной форме.

Младшие десятичные разряды, выходящие за пределы данного формата, при преобразовании отбрасываются. Если количество целых цифр числа больше, нежели заданный для преобразования формат, выдается сообщение об ошибке.

Рассмотрим пример:

```
:A =12.196
:CONVERT A TO A⊙, (# # #)
:PRINT A⊙
012
:C0NVERT A TO A⊙, ( + # # # # . # # # #)
:PRINT  A⊙
+0012.1960
:CONVERT  A  TO  A⊙, (-#./\/\/\)
: PRINT  A⊙
1.2E+01
:CONVERT -A/100 TO A⊙, (-# . # /\/\/\)
:PRINT A⊙
-1.2E-01
:—
```

Если при преобразовании в заданный формат необходимо учесть значение отбрасываемых десятичных разрядов, то не обязательно пользоваться функцией округления ROUND. Достаточно просто добавить к числу половину разряда, следующего за отбрасываемым.

$0.5 \cdot 10^{-(n+1)}$

где n — число знаков после запятой в формате,

Например (продолжение предыдущего примера)

```
:CONVERT A TO A⊙,(# # . # #)
:PRINT A⊙
12.19
:CONVERT A+.005 TO A⊙,(# # . # #)
:PRINT A⊙
12.20
:—
```

13.7. Упаковка и распаковка числовых данных

Формат представления числовых данных в «Искре 226» предусматривает фиксированное число байтов для хранения числовых значений независимо от величины и цифр в числе, т. е. из расчета на максимальную размерность числа. В частности, значения числовых переменных (действительные числа) занимают восемь байт, а значения целочисленных переменных (целые числа) — 2 байта.

В то же время числа, обрабатываемые при решении конкретных задач, могут иметь значительно меньшую размерность. Естественно возникает вопрос, нельзя ли сократить число байтов, отводимое для хранения соответствующих значений. Этот вопрос может иметь принципиальное значение при обработке больших числовых массивов, когда важно по возможности минимизировать требуемый объем оперативной или дисковой памяти.

Когда приходится работать с целыми числами в интервале —7999 до 7999, самой простой возможностью является использование целочисленных переменных. Это позволяет в четыре раза сократить объем памяти для хранения соответствующих значений.

В случае небольшого количества значащих разрядов соответствующие цифры можно хранить в символьной форме. Каждый символ числа будет занимать при этом один байт. Однако при хранении чисел в символьной форме память используется недостаточно эффективно. Дело в том, что байт рассчитан на хранение кодов всех используемых символов — в нем могут храниться коды от 0 до 255, что позволяет закодировать все необходимые символы русского и латинского алфавитов, специальные знаки и команды управления.

Если же использовать байт только для хранения символов Десятичных чисел, то такое количество

кодов является излишним. Для кодирования десятичных цифр с избытком хватает половины байта — в каждом полубайте может храниться код от 0 до F, т. е. от 0 до 15. С учетом этого в «Искре 226» имеется специальный упакованный формат представления чисел, в соответствии с которым в каждый байт записывается по две цифры от 0 до 9, которые интерпретируются как цифры десятичного числа (так называемый десятичноупакованный формат). Это позволяет существенно уменьшить объем памяти, необходимый для хранения массивов чисел с небольшим числом значащих разрядов.

Для преобразования чисел из числовой формы в десятично-упакованный формат в Бейсике используется оператор PACK, а для обратного преобразования UNPACK. При этом для хранения десятичноупакованного формата используются обычные символьные переменные. Следует только помнить, что при попытке вывода их значений, например с помощью оператора PRINT, байты этих переменных будут интерпретироваться как обычные шестнадцатеричные коды в соответствии с кодом КОИ-8.

Общая форма этих операторов PACK и UNPACK

$$\text{PACK}(<\text{формат}>)\left\{\begin{array}{l}<\text{символьная переменная}>\\<\text{обозначение символьного массива}>\end{array}\right\}\text{FROM}\left\{\begin{array}{l}<\text{а.в.}>\\<\text{обозначение числового массива}>\end{array}\right\}[,...]$$

$$\text{UNPACK}(<\text{формат}>)\left\{\begin{array}{l}<\text{символьная переменная}>\\<\text{обозначение символьного массива}>\end{array}\right\}\text{TO}\left\{\begin{array}{l}<\text{числовая переменная}>\\<\text{обозначение числового массива}>\end{array}\right\}[,...]$$

где

$$<\text{формат}> = [\left\{\begin{array}{l}+\\-\end{array}\right\}]\ \ [\#\,\#...\#]\ \ [.]\ \ [\#\,\#...\#]\ \ [/\backslash\backslash\backslash\backslash]$$

Формат в операциях упаковки и распаковки интерпретируется так же, как и при преобразовании данных с помощью оператора CONVERT (см. 13.6), а числа упаковываются по следующим правилам:

каждые два разряда числа упаковываются в один байт, общее число байтов с разрядами числа определяется числом символов # в формате;

если в формате задан знак числа, то он упаковывается в половину байта;

десятичная точка не включается в десятичноупакованное представление (поэтому при распаковке необходимо вновь задавать формат числа);

если в формате задана экспоненциальная форма представления числа, то для записи порядка используется один байт.

Эти правила позволяют точно рассчитать число байтов символьной переменной, необходимое для хранения результата упаковки чисел заданного формата. Например, для записи числа в приведенных ниже форматах требуется:

#-2 байта;
#-2 байта;
+# # # . # # # - 3 байта;
. # # /\\\\ - 4байта.

Как видно из общей записи, с помощью одного оператора можно задать упаковку и распаковку сразу для всех числовых переменных и массивов, перечисленных в списке. При этом упакованные значения записываются вплотную друг к другу. Скажем, для упаковки чисел с помощью оператора

```
PACK  (# # # #)  A⊙  FROM  X1, X2,  X3,  X4
```

достаточно 8 байт переменной A⊙.

П р и м е р 13.9

Следующая программа упаковывает и затем распаковывает числа. Предполагается, что числа имеют величину от —999 до +999, а дробных частей нет или они не существенны.

```
10  REM  ПРОГРАММА С УПАКОВКОЙ И РАСПАКОВКОЙ ЧИСЕЛ
20  DIM  A⊙ (10),X(40),Y(30),Z(10)
. . .
100  REM  НЕОБХОДИМО УПАКОВАТЬ ЧИСЛА
110  PACK  (+# # #)A⊙  ( )FROM  X( ),Y( ),Z( )
. . .
```

```
200  REM  НЕОБХОДИМО РАСПАКОВАТЬ ЧИСЛА
210  UNPACK  ( + # # #)A⊙  ( )TO X( ),Y( ),Z ( )
```

Глава 14. ОПЕРАЦИИ С ДВОИЧНЫМИ ЧИСЛАМИ. ЛОГИЧЕСКИЕ ОПЕРАЦИИ

До сих пор при изучении операторов языка описывались операции над двумя типами данных: числовыми и символьными. Числовые данные представлялись в десятичной системе счисления, и для хранения результатов выполнения арифметических операций и вычисления числовых функций использовались числовые переменные. Символьные переменные использовались для хранения текстовых констант (шестнадцатеричных кодов). Однако двумя типами данных возможности языка Бейсик по представлению данных не исчерпываются. В символьных переменных, как отмечалось в гл. 13, записывается некоторая информация в двоичном виде. Эта информация может рассматриваться по-разному. Различие в типах данных будет определяться в этом случае видом операций, проводимых над содержимым символьной переменной. В зависимости от вида операции данные, хранимые в символьной переменной, могут рассматриваться как строка текста, последовательность шестнадцатеричных кодов, двоичное число или последовательность двоичных чисел, последовательность двоичных нулей и единиц.

14.1. Операции с двоичными числами

Овладение операциями над двоичными числами и двоичном формой представления необходимы для дальнейшего изучения ряда операторов языка Бейсик достаточно общего назначения таких, как операторы поиска и сортировки. В этих операторах, как и в ряде других, используется двоичная форма представления результатов.

Двоичная форма представления числа является наиболее компактной формой записи числа в памяти машины как во внутренней (оперативной), так и во внешней памяти на дисках. Действительно, в одном байте символьной переменной можно записать число от 0 до 255, в двух байтах — от 0 до 65535 и так далее. Поэтому для хранения небольших по абсолютной величине чисел иногда выгодно использовать двоичное представление чисел. В языке предусмотрен лишь один вид операций над двоичными числами—операция двоичного сложения. Другие арифметические операции над двоичными числами при необходимости могут быть реализованы с помощью операторов двоичного сложения и логических операторов.

Аргументами в операциях над двоичными числами являются символьные переменные и двоичные константы. Содержимое символьных переменных можно рассматривать как двоичные числа с большим количеством двоичных разрядов. При этом количество разрядов будет кратно произведению числа восемь на длину символьной переменной в байтах. В другом случае содержимое символьных переменных рассматривается как последовательность восьмиразрядных двоичных чисел, где каждое двоичное число занимает один байт символьной переменной.

Двоичная константа в операциях двоичного сложения задается шестнадцатеричным числом, записывающимся двумя шестнадцатеричными цифрами. Вообще для операций ввода и присваивания двоичного содержимого символьной переменной нет необходимости вводить специальный вид двоичной константы. Достаточно использовать шестнадцатеричную функцию HEX. Заметим, что задаваемая разрядность всегда будет кратна восьми разрядам, так как один байт (единица измерения памяти машины) состоит из восьми двоичных разрядов. Каждый разряд байта принимает значение нуля или единицы; таким образом число, записанное в байте в двоичном виде, равняется

$$c_7\cdot 2^7 + c_6\cdot 2^6 + c_5\cdot 2^5 + c_4\cdot 2^4 + c_3\cdot 2^3 + c_2\cdot 2^2 + c_1\cdot 2^1 + c_0\cdot 2^0,$$

где $c_i = (0,1)$, $0 \le i \le 7$.

Таким образом, в одном байте можно записать в двоичном виде число от нуля (все разряды байта $c_i = 0$) до 255 (все разряды байта $c_i = 1$). Для задания двоичного содержимого одного байта достаточно использовать шестнадцатеричную функцию с двумя шестнадцатеричными цифрами. Например, число 0 задается функцией HEX (00), число 255 функцией HEX(FF), число 16 — функцией HEX (10). Для задания двоичного числа со значениями в диапазоне от 0 до 2^{16}—1 необходимо использовать шестнадцатеричную функцию с четырьмя шестнадцатеричными цифрами, а для хранения такого числа нужно два байта символьной переменной. При увеличении диапазона значений двоичного числа до 2^{24}—1 необходимо использовать для задания значений шестнадцатеричную функцию с шестью цифрами

и соответственно три байта символьной переменной для хранения содержимого и так далее.

Операция двоичного сложения выполняется оператором ADD. Содержимое аргументов этого оператора — двух символьных переменных или одной символьной переменной и двухразрядного шестнадцатеричного числа (восьмиразрядного двоичного числа) — складывается по правилам сложения двоичных чисел, а результат сложения заносится в содержимое первого аргумента оператора ADD. Пусть, например, содержимое переменной B⊙ длиной один байт складывается с числом 100, заданным шестнадцатеричным числом HEX (64). Первоначально содержимое переменной B⊙ было установлено равным числу 60.

П р и м е р 14.1. Двоичное сложение двух чисел 60 и 100.

```
10  DIM B⊙1
20  B⊙=HEX(3C):REM ЧИСЛО 60 В ШЕСТНАДЦАТИРИЧНОМ ВИДЕ
30  ADD(B⊙,64): REM СЛОЖЕНИЕ С ЧИСЛОМ 100
40  HEXPRINT B⊙
```

Естественно ожидать, что результат операции сложения (содержимое переменной B⊙), будет равен числу 160 или в шестнадцатеричном виде числу HEX (A0). Результат сложения, конечно, не изменяется от вида представления числа (двоичного, десятичного или шестнадцатеричного). Для наглядности проиллюстрируем операцию двоичного сложения, выполненную в предыдущем примере. Содержимое переменной B⊙ до операции сложения, представляющее собой число 100, в двоичном виде

01100100

Число 60 в двоичном виде

00111100

Результат сложения, записываемый в переменную B⊙, равен

$$\begin{array}{r} 01100100 \\ + \\ 00111100 \\ \hline 10100000 \end{array}$$

Примечание. При сложении разрядов, содержащих две единицы, содержимое соответствующего разряда результата становится равным нулю, и единица переносится в следующий разряд.

Используя оператор ADD можно складывать двоичные числа двумя способами. При сложении первым способом перенос между байтами отсутствует, т. е. если при сложении двух старших разрядов байта вырабатывается перенос, то он теряется. При этом способе сложение двух символьных переменных осуществляется побайтно.

Второй способ предполагает сложение двух двоичных чисел с переносом между байтами. Границы между байтами игнорируются, а разряды двоичного числа рассматриваются как в позиционной двоичной системе счисления, начиная со старшего разряда первого байта символьной переменной. Этот разряд — самый старший двоичный разряд числа, а самым младшим разрядом числа является младший разряд последнего байта символьной переменной.

Оператор двоичного сложения имеет следующий общий вид:

$$\mathrm{ADD}[\mathrm{C}]\left(< \text{символьная переменная 1} >, \left\{ \begin{array}{l} \mathrm{hh} \\ < \text{символьная переменная 2} > \end{array} \right\}\right)$$

где

параметр C — определяет способ сложения. При отсутствии параметра C сложение происходит без переноса между байтами. Если параметр C присутствует, сложение происходит с переносом между байтами;

<символьная переменная 1>—служит для хранения первого аргумента до сложения и результата операции после сложения;

Второй аргумент операции сложения может быть записан в двух формах:

hh — две шестнадцатеричные цифры, определяющие двоичное число, складываемое с содержимым первого аргумента. Если параметр C отсутствует, hh складывается с содержимым каждого байта символьной переменной. Если параметр C присутствует, hh складывается только с содержимым последнего байта символьной переменной;

<символьная переменная 2>—содержимое этой переменной складывается с содержимым переменной 1. Если длина символьных переменных различна, то символьная переменная с меньшей длиной выравнивается по длине символьной переменной с большей длиной, т. е. складываются сначала последние байты переменных, потом предпоследние и так далее до первого байта одной из переменных.

П р и м е р 14.2. Пусть содержимое переменной A⊙ определено следующим образом:

```
10  DIM A⊙2
20  A⊙ = HEX (0123)
```

После выполнения операции сложения содержимое переменной A⊙ изменится следующим образом:

```
:ADD (A⊙,02) : HEXPRINT A⊙
0325
:ADDC (A⊙,02) : HEXPRINT A⊙
0125
:ADD (A⊙, FF) : HEXPRINT A⊙
0022
:ADDC (A⊙, FF) : HEXPRINT A⊙
0222
```

П р и м е р 14.3. Рассмотрим результаты выполнения программы, в которой складывается содержимое двух символьных переменных.

```
10  DIM A⊙3, B⊙2, C⊙3
20  B⊙=HEX(01FF)
30  STR(A⊙,1,2)=HEX(0001) : REM СОДЕРЖИМОЕ A⊙ РАВНО HEX (000120)
40  C⊙=A⊙ : ADD (C⊙, B⊙). : HEXPRINT C⊙
50  C⊙ = A⊙ : ADDC(C⊙, B⊙) : HEXPRINT C⊙
60  C⊙=A⊙ : ADD(STR(C⊙l, 2), B⊙) : HEXPRINT C⊙
:RUN
00020F
00030F
01F020
01F020
```

14.2. Преобразование двоичных чисел в десятичные

Для преобразования десятичного числа в двоичное число используется оператор BIN. Этот оператор преобразует целую часть арифметического выражения в двоичное число и записывает это число в двоичном представлении в первом байте или в первых двух байтах символьной переменной. Общая форма оператора имеет следующий вид:

BIN (<символьная переменная> [,2]) = <а.в.>

где

параметр 2 — определяет число байтов, содержащих преобразованное арифметическое выражение.

При отсутствии второго параметра арифметическое выражение преобразуется в двоичное число, записываемое в первом байте символьной переменной. Если второй параметр присутствует, двоичное число, полученное в результате преобразования, записывается в первом и втором байтах символьной переменной.

При использовании оператора BIN без второго параметра Двоичное число записывается восемью двоичными разрядами. Поэтому значение арифметического выражения (А) должно быть в пределах

$0 \le A \le 255$

При использовании оператора BIN с параметром 2 двоичное число записывается шестнадцатью двоичными разрядами. В этом случае значение арифметического выражения должно быть в пределах

$0 \le A \le 65535$

Если значение арифметического выражения в первом или втором случае превысит соответствующие пределы, то при выполнении оператора возникнет ошибка.

Операция преобразования значений числовой переменной E% в двоичное содержимое первого байта переменной B⊙ может быть записана, например, следующим образом:

```
BIN(B⊙) = E%
```

Операция преобразования значения переменной К в двоичное содержимое первых двух байтов переменной C⊙ может быть записана следующим образом:

```
BIN(C⊙, 2)=К
```

П р и м е р 14.4. При выполнении следующей программы распечатываются результаты реализации оператора BIN в различных вариантах.

```
10  DIM /\⊙2
20  INIT(00) A⊙
30  BIN(A⊙)= 1 : HEXPRINT A⊙
40  BIN(A⊙) = 32 : HEXPRINT A⊙
50  BIN(A⊙)= 47 : HEXPRINT A⊙
60  BIN(STR(A⊙, 2, 1)) = 255 : HEXPRINT A⊙
70  BIN(A⊙, 2)= 1 : HEXPRINT A⊙
80  BIN(A⊙, 2)= 33 : HEXPRINT A⊙
90  BIN(A⊙, 2)= 256 : HEXPRINT A⊙
100  BIN(A⊙, 2)= 65534 : HEXPRINT A⊙
:RUN
0100
2000
2F00
2FFF
0001
0021
0100
FFFE
```

Для преобразования чисел из двоичного представления в десятичное представление в языке Бейсик предусмотрена функция VAL, которая является числовой функцией символьного аргумента. Она преобразует двоичное значение содержимого первого байта или первых двух байтов символьной переменной, или константы в десятичное число. Функция VAL выполняет преобразование, обратное преобразованию, выполняемому оператором BIN. Общая форма функции имеет следующий вид:

$$\text{VAL}\left(\left\{\begin{array}{l}< \text{символьная переменная} >\\ < \text{символьная переменная} >\end{array}\right\}[,2]\right)$$

Например, операция преобразования двоичного содержимого первого байта переменной B⊙ в десятичное содержимое переменной E% может быть записана так:

```
E%=VAL(B⊙)
```

Преобразование двоичного содержимого двух первых байтов переменной C⊙ в десятичное содержимое переменной K выглядит следующим образом:

```
K=VAL(C⊙, 2)
```

Выполнение следующих операторов приведет к таким результатам:

```
:A⊙=HEX(00): PRINT VAL(A⊙)
0
:A⊙=HEX(FF): PRINT VAL(A⊙)
255
:A⊙=HEX(0001): PRINT VAL(A⊙, 2)
1
:A⊙=HEX(0200): PRINT VAL(A⊙, 2)
512
```

В заключение рассмотрим связь операций преобразования из двоичной формы представления и обратно с параметром 2 и без него. Функция VAL с двумя параметрами выражается через функцию VAL с одним параметром следующим образом:

```
VAL(X⊙, 2)=VAL(X⊙)*256 + VAL(STR(X⊙, 2))
```

Результат выполнения оператора

```
BIN(X⊙, 2)=X
```

аналогичен результату выполнения операторов

```
BIN(X⊙)=X/256
```

и

```
BIN (STR (X⊙, 2)) =X — INT (X/256) *256
```

14.3. Логические операции

Одним из важных типов операций над содержимым символьных переменных являются логические операции. Аргументами в логических операциях служат символьные переменные, содержимое которых рассматривается как последовательность нулей и единиц, записанных в двоичных разрядах байтов символьных переменных. Результаты логической операции записываются в символьную переменную, значения каждого двоичного разряда которой определяются логической операцией между соответствующими разрядами символьных переменных. При этом результат логической операции между значениями соответствующих разрядов определяется только содержимым этих разрядов и не зависит от содержимого других разрядов. Таким образом, в логической операции используются два аргумента. Каждый из аргументов представляет собой содержимое одного двоичного разряда символьной переменной или константы. В логических операциях могут использоваться двоичные константы, задаваемые шестнадцатеричным числом. Логическая операция проводится отдельно над каждым разрядом одной символьной переменной (аргументом 1) и соответствующим разрядом другой символьной переменной пли константы (аргументом 2). Для соответствующих байтов символьных переменных логическая операция проводится отдельно над содержимым первых разрядов байта, вторых разрядов байта и так далее.

Поскольку в логической операции принимают участие два аргумента с возможными значениями нуль или единица, количество возможных комбинаций значений аргументов равно четырем. Действительно, пусть X_1 — значение первого аргумента, а X_2 — значение второго аргумента. Тогда все возможные комбинации значений двух аргументов могут быть представлены в табл. 14.1.

Результатом логической операции является одно из двух значений: нуль или единица. Любую логическую операцию можно задать в табличном виде, записывая четыре значения результата для каждой комбинации аргументов (см. табл. 14.2).

Таблица 14.1

X_1	X_2
0	0
0	1
1	0
1	1

Таблица 14.2

X_1	X_2	Операция „ИЛИ“
0	0	0
0	1	1
1	0	1
1	1	1

Операция «ИЛИ», заданная таблицей, выполняет логическое объединение значений аргументов. Результат этой операции равен нулю только в том случае, когда значение обоих аргументов равно нулю.

Рассмотрим еще две логические операции, которые наряду с операцией «ИЛИ» являются наиболее употребимыми. Одна из них — это операция «И», результаты которой задаются табл. 14.3.

Операция «И» выполняет логическое пересечение значений аргументов, так как ее значение равно единице, тогда и только тогда, когда значения обоих аргументов равны единице.

Операция «Исключающее ИЛИ» приводит к следующим результатам (см. табл. 14.4).

Операция «Исключающее ИЛИ» выполняет объединение двух различных значений аргументов. Действительно, результат

Таблица 14.3

X_1	X_2	Операция „И“
0	0	0
0	1	0
1	0	0
1	1	1

Таблица 14.4

X1	X2	Операция Исключающее „ИЛИ“
0	0	0
0	1	1
1	0	1
1	1	0

равен единице, когда значения аргументов различны, и равен нулю, когда значения аргументов совпадают. Эта операция называется также «Сложение по модулю 2», так как ее результат совпадает с результатом арифметической операции сложения двух одноразрядных двоичных чисел в одном разряде. Для выполнения логических операций в языке машины предусмотрены соответствующие операторы следующего вида:

$$\left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{XOR}\end{matrix}\right\}\left(\text{< символьная переменная 1 >},\left\{\begin{matrix}\text{hh}\\ \text{< символьная переменная 2 >}\end{matrix}\right\}\right)$$

OR — наименование оператора, выполняющего операцию "ИЛИ";
AND — наименование оператора, выполняющего операцию "И":
XOR — наименование оператора, выполняющего операцию "Исключающее ИЛИ";
hh — две шестнадцатеричные цифры, задающие константу в качестве второго аргумента.

В операторах в качестве первого аргумента используются двоичные разряды <символьной переменной 1> начиная с первого байта этой переменной. В качестве второго аргумента используются соответствующие разряды либо <символьной переменной 2>, начиная с первого байта этой переменной, либо разряды, определенные константой hh. В последнем случае логическая операция производится с содержимым разрядов всех байтов <символьной переменной 1>. Результаты выполнения оператора записываются в соответствующие разряды <символьной переменной 1>.

Рассмотрим на примере результаты выполнения описанных выше логических операторов. Результаты записываются в переменную A⊙, а программа распечатывает шестнадцатеричное значение этой переменной.

```
10  DIM A⊙2, B⊙2, C⊙2
20  C⊙=HEX(4145) : REM НАЧАЛЬНЫЕ ЗНАЧЕНИЯ
АРГУМЕНТОВ ПЕРЕМЕННОЙ 1
30  B⊙ = HEX(2185) : REM НАЧАЛЬНЫЕ ЗНАЧЕНИЯ
АРГУМЕНТОВ ПЕРЕМЕННОЙ 2
40  A⊙=C⊙ : AND(A⊙, B⊙) : HEXPRINT A⊙
50  A⊙=C⊙ :  OR(A⊙, B⊙)  :  HEXPRINT A⊙
60  A⊙=C⊙ :  XOR(A⊙, B⊙)  :  HEXPRINT A⊙
:RUN
0105
6165
60C0
```

Выполнение следующих операций приведет к таким результатам.

```
:AND(A⊙00) :  HEXPRINT  A⊙
0000
```

Операция «И» выполняется с аргументом, значения всех разрядов которого равны единице. Поэтому значения всех двоичных разрядов результата равны единице безотносительно первоначального содержимого переменной A⊙.

```
:OR(A⊙FF)  :  HEXPRINT  A⊙
0000
```

Операция «ИЛИ» выполняется с аргументом, значения всех; разрядов которого равны единице. Поэтому значения всех разрядов результата равны единице безотносительно содержимого переменной A⊙.

```
:XOR(A⊙, A⊙)  :  HEXPRINT  A⊙
0000
```

Операция «ИЛИ» выполняется с аргументами, содержащими одинаковые значения. Поэтому результат этой операции равен нулю при любых значениях переменной A⊙.

```
:A⊙ = HEX(970F)  :  XOR(A⊙, FF)  :  HEXPRINT  A⊙
68F0
```

Операция «Исключающее ИЛИ» выполняется с аргументом, значения всех двоичных разрядов которого равны единице. Поэтому в результате выполнения такой операции первоначальные значения разрядов переменной A⊙ заменяются обратными, т. е. нули заменяются на единицы, а единицы на нули. Заметим, что результат этой операции является дополнением первоначального содержимого переменной A⊙ до числа 15 в каждом шестнадцатеричном разряде. Поэтому сумма первоначального содержимого переменной A⊙ и результата выполнения оператора XOR (A⊙,FF) всегда будет равна HEX (FFFF) при любом начальном) содержимом переменной A⊙. Например,

```
:C⊙ = A : XOR(A⊙,FF) : ADD(A⊙,C⊙) : HEXPRINT A⊙
FFFF
```

Логические операторы могут быть использованы для различных преобразований содержимого символьной переменной. Для выделения отдельных разрядов переменной достаточно использовать оператор AND, в котором единичные значения второго аргумента задают выделяемые разряды. Для инвертирования содержимого символьной переменной можно использовать оператор XOR со значением единицы во всех разрядах второго аргумента. Для маскирования отдельных разрядов переменной, содержимое которых безразлично для последующих операций сравнения, можно использовать оператор OR со значениями единицы в маскируемых разрядах и так далее.

Кроме приведенных операторов, в языке Бейсик предусмотрен оператор BOOL, позволяющий выполнить любую логическую (булевскую) операцию, включая и рассмотренные выше. Задаваемая логическая операция выполняется над содержимым символьной переменной (аргументом 1) и содержимым символьной переменной или константы (аргументом 2).

Общая форма этого оператора выглядит следующим образом:

$$\text{BOOL x}\left(<\text{символьная переменная 1}>,\left\{\begin{array}{l}\text{hh}\\ <\text{символьная переменная 2}>\end{array}\right\}\right)$$

где

x—шестнадцатеричная цифра от нуля до F, обозначающая соответствующую логическую операцию.

Аргументы в операторе BOOL задаются так же, как и в рассмотренных логических операторах. С помощью оператора BOOL можно выполнить любую из шестнадцати возможных логических операций. Значение x, представляемое в двоичном виде четырьмя разрядами, является результатом выполняемой операции, если задавать комбинации значений аргументов в порядке, приводимом в табл. 14.5.

Таблица 14.5

Обозначение операции x в операторе BOOL	Функция	Значения аргументов и результатов			
		Аргумент 1			
		1	1	0	0
		Аргумент 2			
		1	0	1	0
0	Всегда ложно	0	0	0	0
1	Отрицание дизъюнкции	0	0	0	1
2	Отрицание обратной импликации	0	0	1	0
3	Отрицание аргумента 1	0	0	1	1
4	Отрицание импликации	0	1	0	0
5	Отрицание аргумента 2	0	1	0	1
6	Исключающее ИЛИ	0	1	1	0
7	Связь Шеффера	0	1	1	1
8	И	1	0	0	0
9	Равнозначность	1	0	0	1
A	Не зависит от аргумента 1	1	0	1	0
B	Импликация	1	0	1	1
c	Не зависит от аргумента 2	1	1	0	0
D	Обратная импликация	1	1	0	1
E	ИЛИ	1	1	1	0
F	Всегда истинно	1	1	1	1

Выполнение оператора BOOL может быть проиллюстрировано следующим примером.

Пример 14.5. Рассмотрим результат выполнения оператора при необходимости вычисления логической функции «Связь Шеффера». Программа распечатывает значения переменной A⊙ используемой для хранения результата вычисления функции.

```
10  DIM A⊙2,B⊙2
20  A⊙=HEX(4541) : B⊙=HEX(2185) : REM НАЧАЛЬНЫЕ
ЗНАЧЕНИЯ АРГУМЕНТОВ
30  B00L 7 (A⊙,B⊙)
40  HEXPRINT A⊙
:RUN
FEFA
```

Операторы BOOL 8 и AND, BOOL E и OR, BOOL 6 и XOR соответственно эквивалентны друг другу.

14.4. Оператор циклического сдвига

Содержимое символьных переменных можно рассматривать как последовательность нулей и единиц, записанных в двоичных разрядах каждого байта символьной переменной, или как такую же последовательность во всей символьной переменной. В последнем случае границы между байтами игнорируются. Операцией сдвига двоичной последовательности на один двоичный разряд влево или вправо является операция, при которой содержимое разряда 2^{i+1} при сдвиге влево или разряда 2^{i-1} при сдвиге вправо становится равным исходному содержимому разряда 2^i двоичной последовательности. Например, пусть исходная последовательность равна:

разряды	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
значение	0	1	0	1	1	0	1	0

При сдвиге на один разряд влево содержимое разряда будет равняться:

разряды	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
значение	1	0	1	1	0	1	0	0

При сдвиге на один разряд вправо получим:

разряды	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
значение	0	0	1	0	1	1	0	1

Операция сдвига имеет простую интерпретацию, если рассматривать содержимое двоичной последовательности как некоторое двоичное число. Тогда операция сдвига на k разрядов влево означает умножение исходного числа на 2^k, а операция сдвига на k разрядов вправо обозначает операцию деления исходного числа на 2^{κ}. Действительно, для приведенного выше примера исходное значение двоичного числа в байте равно 90. Сдвиг на один разряд влево приводит к значению числа, равному 180 = =90·2. Сдвиг на один разряд вправо приводит к значению числа, равному 45 = 90/2.

Операция простого сдвига при фиксированной разрядной сетке приводит к потере старших при сдвиге влево или младших при сдвиге вправо разрядов исходного числа. Операция циклического сдвига отличается от простого сдвига тем, что содержимое сдвигаемых разрядов, не умещающихся в исходной разрядной сетке, не теряется. Это содержимое замещает содержимое самых младших при сдвиге влево или самых старших при сдвиге вправо разрядов. Например, пусть исходное двоичное число, записанное в одном байте, равно:

разряды	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
значение	1	0	1	1	0	0	1	0

При циклическом сдвиге влево на один разряд исходное содержимое разряда 2^7 становится содержимым разряда 2°:

разряды	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
значение	0	1	1	0	1	0	1	1

При циклическом сдвиге вправо на один разряд исходное содержимое разряда 2°2 становится содержимым разряда 2^7:

разряды	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
значение	1	1	0	1	1	0	1	0

Оператор ROTATE предназначен для циклического сдвига содержимого символьной переменной на 1—8 разрядов. Общая форма оператора имеет следующий вид:

ROTATE [C] (<символьная переменная>, <а.в.>)

где

<символьная переменная> — содержит исходную и результирующую последовательность;
параметр С — определяет вид сдвига;
<а.в.> — определяет кратность и направление сдвига.

Отсутствие параметра С обозначает проведение операции циклического сдвига с содержимым каждого байта символьной переменной. Если в операторе задан параметр С, границы между байтами игнорируются и содержимое всей символьной переменной циклически сдвигается. Арифметическое

выражение, определяющее кратность и направление сдвига, должно находиться в диапазоне:

1≤|<а.в.>|≤8

Направление сдвига определяется знаком арифметического выражения. При положительном значении выражения осуществляется циклический сдвиг влево, при отрицательном значении выражения — вправо.

П р и м е р 14.6. В результате выполнения программы распечатываются следующие результаты:

```
10  DIM A⊙2
20  A⊙=HEX (C33C)  :   ROTATE (A⊙, 1)   :  HEXPRINT  A⊙
30  A⊙=HEX (C33C)  :   ROTATE (A⊙, -2)   :  HEXPRINT  A⊙
40  A⊙=HEX (C33C)  :   ROTATE  C (A⊙, 1)   :  HEXPRINT  A⊙
50  A⊙=HEX (C33C)  :   ROTATE  C (A⊙, 4)   :  HEXPRINT  A⊙
60  A⊙=HEX (C33C)  :   ROTATE  C (A⊙, -4)   :  HEXPRINT  A⊙
70  A⊙=HEX (C33C)  :  ROTATE  C (A⊙, 8)   :  HEXPRINT  A⊙
80  A⊙=HEX (C33C)  :   ROTATE  C (A⊙, -8)   :  HEXPRINT  A⊙
90  A⊙=HEX (C33C)  :   ROTATE  C (A⊙, 0)   :  HEXPRINT  A⊙
:RUN

8778
F00F
8679
33CC
CC33
3CC3
C33C
```

П р и м е р 14.7. Следующие операторы выполняют деление двоичного содержимого переменной B⊙ на 2.

```
. . .
100  ROTATE   C (B⊙, —1)
110  AND (STR (B⊙, 1, 1), 7F)
. . .
```

Оператор в строке 100 производит циклический сдвиг всего содержимого переменной B⊙ на один разряд, Оператор в строке 110 обнуляет самый старший разряд переменной B⊙, в которой после циклического сдвига попадет значение самого младшего разряда.

П р и м е р 14.9. В программах редактирования текста часто возникает необходимость выполнения операций удаления или вставки символа в определенной позиции символьной переменной. Следующие подпрограммы выполняют эти операции с содержимым переменной T⊙. Вставляемый символ и позиция вставки или удаления являются параметрами подпрограмм.

```
10   DIM T⊙253, A⊙1
. . .
100   REM ВСТАВКА СИМВОЛА A⊙ В ПОЗИЦИЮ X
110   DEFFN' 100 (A⊙, X)
120   ROTATE C (STR (T⊙, X), -8)
130   STR (T⊙, X,1)=A⊙
140   RETURN
150   REM УДАЛЕНИЕ СИМВОЛА ИЗ ПОЗИЦИИ X
160   DEFEN' 101(X)
170   STR (T⊙, X, 1) = HEX (20)
180   ROTATE C (STR (T⊙, X), 8)
190   RETURN
```

Глава 15. ОПЕРАЦИИ ПОИСКА, ЗАМЕНЫ, КОПИРОВАНИЯ И ПЕРЕКОДИРОВКИ СИМВОЛЬНЫХ ДАННЫХ

В этой главе рассматриваются средства языка Бейсик, предназначенные для выполнения операций поиска и замены содержимого символьных переменных и символьных массивов. Описываемые

операторы обладают очень широкими возможностями обработки символьных данных. Поиск данных в содержимом символьной переменной можно проводить по самым различным условиям. При замене содержимого символьной переменной можно изменять как ее отдельные символы, так и любые строки символов в ней и т. д. Отличительной особенностью ряда описываемых операторов является возможность выполнения групповых операций, в результате которых одна и та же элементарная операция выполняется многократно над всем содержимым символьной переменной. Так, например, в результате исполнения одного оператора можно найти или заменить сразу все искомые значения в символьной переменной. Применение таких операторов позволяет во многих случаях существенно повысить эффективность составления алгоритмов и программ работы с символьными данными. Использование этих операторов обеспечивает и существенный выигрыш во времени выполнения программы. Дело в том, что вследствие интерпретирующего режима выполнения операторов языка Бейсик всегда предпочтительнее программировать нужную операцию возможно меньшим количеством операторов. При этом использование одной групповой операции, выполняемой одним оператором над всем содержимым символьной переменной, эквивалентно целой группе операторов, многократно выполняющих такие же операции с отдельными элементами символьной переменной.

15.1. Операции поиска в символьных переменных и массивах

При решении ряда задач бывает необходимо использовать операцию поиска какой-либо части содержимого символьной переменной. Например, необходимо найти отдельное слово, часть слова, букву, цифру или другой символ в символьной переменной или символьном массиве, что означает определить положение первого символа символьной переменной, начиная с которого в этой переменной записаны искомые данные. Таким образом, в операции поиска результатом является номер байта символьной переменной, начиная с которого в этой переменной содержатся данные, удовлетворяющие заданному условию. Задаваемое при поиске условие не обязательно является отношением равенства, когда идет поиск данных, равных поисковому значению. Отношение, задаваемое при поиске, может принимать значения «равно», «меньше», «больше» и комбинации этих значений, таких, как «больше или равно», «меньше или равно» и «не равно». При поиске можно задавать такие же отношения сравнения, как и при сравнении числовых данных. Любой символ текста хранится в памяти машины в виде определенного шестнадцатеричного кода (приложение 1). В операции поиска эти коды, или, точнее говоря, их числовые значения, служат для сравнения искомого значения с содержимым поисковой переменной. Поэтому в операциях поиска можно искать не только цепочку текстовых символов, удовлетворяющих заданному отношению, но и любую цепочку шестнадцатеричных кодов, где каждый код может принимать значение от HEX (00) до HEX(FF).

Для выполнения операций поиска в символьных переменных в языке Бейсик предусмотрен оператор поиска и функция поиска, применение которых описано в настоящем разделе. Поиск проводится в содержимом задаваемой в функции или операторе символьной переменной, называемой далее поисковой переменной, Поисковой переменной может быть символьная переменная, символьный массив или его часть. Если поиск проводится в содержимом символьного массива, то в качестве поисковой переменной задается обозначение этого символьного массива. При поиске в массиве границы между элементами массива игнорируются, а весь массив рассматривается как одна строка символов. Содержимое этой строки состоит из содержимого элементов массива, расположенных в порядке возрастания номеров их индексов.

Если поиск нужно проводить не во всем содержимом массива, а только в его части, для выделения части поисковой переменной используется функция STR с параметрами, выделяющими нужную часть массива. Содержимое этой части рассматривается как часть содержимого одной строки символов всего массива, представленной так же, как и в случае поиска по всему массиву.

Функция поиска POS является числовой функцией символьного аргумента. Функция принимает значение первого номера байта в поисковой символьной переменной, в котором удовлетворяется заданное отношение. Отношение задается в скобках функции POS в следующем виде:

<поисковая переменная> <символ отношения> <искомая величина>

где символ отношения задается следующими знаками:

<= — меньше или равно;
< — меньше;
= — равно;

> — больше;
>= — больше или равно;
<> — не равно.

Искомой величиной в функции POS является один символ, или, точнее, шестнадцатеричный код. Искомая величина может задаваться содержимым первого байта символьной переменной или символьной константы, или просто одним шестнадцатеричный кодом. Например,

```
1) X=POS (A⊙="M")
2) X=POS (C⊙<="9")
3) X=POS (B⊙<>E  )
4) X=POS (M⊙=20)
5) X=POS (E⊙( ) = ".")
6) X=POS (STR (K⊙( ), 10)<  >HEX (20))
```

В первом случае в переменной A⊙ ведется поиск номера байт, в котором записана буква M, и найденное значение номера присваивается переменной X. Во втором случае переменной X присваивается номер байта в переменной C⊙, значение которого имеет код меньший или равный коду HEX(39). В третьем случае переменной X присваивается номер байта в переменной B⊙, в котором записан код, не совпадающий с кодом первого байта переменной E⊙. В четвертом случае переменной X присваивается номер байта в переменной M⊙, в котором записан код пробела HEX (20). В пятом случае переменной X присваивается номер 5айта в массиве E⊙(), в котором записан символ точки. В последнем случае переменной X присваивается номер первого байт в той части массива K⊙(), где отсутствует код пробела. В последнем случае код пробела задан шестнадцатеричной функцией, что аналогично заданию двух шестнадцатеричных цифр, как в четвертом примере.

Поисковая переменная при вычислении значения функции POS просматривается, начиная с первого байта, и поиск прекращается после нахождения первого же номера байта, удовлетворяющего заданному условию. Если поиск заканчивается безрезультатно, функция POS принимает нулевое значение.

П р и м е р 15.1. В приводимом фрагменте программы в символьную переменную T⊙ вводятся числа целые и дробные. Пусть при вводе дробная часть числа может отделяться от целой части как точкой, так и запятой. В то же время число вида 21,000... с нулями в дробной части считается целым. В программе необходимо определить тип числа и далее перейти к соответствующей части программы.

```
10  DIM  T⊙15
. . .
100   INPUT  "ЧИСЛО",   T⊙
110  X=POS (T⊙ =   ".")
120   IF X<>0  THEN   150:  REM  РАЗДЕЛИТЕЛЬ-ТОЧКА
130   X   =POS (T⊙ =   ",")
140   IF X = 0 THEN 500: REM РАЗДЕЛИТЕЛЬ-ЗАПЯТАЯ
150   IF X=15 THEN 500: REM НЕТ ДРОБНОЙ ЧАСТИ
160   IF P0S (STR (T⊙, X+l) > "0") = 0 THEN 500:
REM НУЛИ В  ДРОБНОЙ ЧАСТИ
170   REM ОБРАБОТКА ДРОБНОГО ЧИСЛА
. . .
500 REM ОБРАБОТКА ЦЕЛОГО ЧИСЛА
```

П р и м е р 15.2. Часто в программе бывает необходимо проверить содержимое одного байта переменной на принадлежность символа, записанного в нем, группе некоторых символов, для каждого из которых нужно выполнить соответствующую обработку. Приводимый фрагмент программы иллюстрирует использование функции POS в этих целях. В этом фрагменте поисковая переменная задается списком символов, а искомая величина — переменной.

```
10 DIM A⊙1 :REM ПЕРЕМЕННАЯ ДЛЯ АНАЛИЗИРУЕМОГО СИМВОЛА
. . .
200 REM  ВЫДЕЛЕНИЕ СИМВОЛОВ  +,  - ,  * ,  / , /\
210 ON P0S (" + - * / /\ "=A⊙) GOTO 300,400,500,600,700
220 REM В A⊙ НЕТ ВЫДЕЛЕННЫХ СИМВОЛОВ
```

Используя функцию POS, составим программу, моделирующую обычный калькулятор. Калькулятор выполняет арифметические операции над значением сумматора и вводимым значением и помещает результат в сумматор. Перед записью числа вводится знак операции. Если знак операции не задан, то содержимое сумматора заменяется введенным числом.

П р и м е р 15.3. Использование калькулятора с сумматором показано на следующем примере:

```
5   REM ВВОД ЗНАКА ОПЕРАЦИИ И ЧИСЛА
10   INPUT"ОПЕРАЦИЯ И ЧИСЛО", A⊙
20   ON POS (" + - * / /\ "=A ) GOTO 50 , 60, 70, 80, 90
25   REM НЕТ ЗНАКА ОПЕРАЦИИ. ЗАМЕНА СОДЕРЖИМОГО СУММАТОРА
30   CONVERT A⊙ TO S
40   PRINT "СУММАТОР = ";S: GOTO 10
50   GOSUB  100 :   S = S+X   :   GOTO  40
60   GOSUB  100 :   S = S-X   :   GOTO  40
70   GOSUB  100 :   S = S*X   :   GOTO  40
80   GOSUB  100 :   S = S/X   :   GOTO  40
90   GOSUB  100 :   S = S/\X   :   GOTO  40
95   REM ПОДПРОГРАММА ПРЕОБРАЗОВАНИЯ СИМВОЛЬНОГО
ПРЕДСТАВЛЕНИЯ ВВЕДЕННОГО ЧИСЛА В ЧИСЛОВОЙ ВИД
100   CONVERT STR (A⊙, 2) TO X : RETURN
```

При выполнении программы получим, например

```
:RUN
? 5
СУММАТОР = 5
? —1
СУММАТОР = 4
?/\3
СУММАТОР = 64
? 0
СУММАТОР = 0
```

и так далее.

Функция POS используется для поиска только одного символа. При вычислении значения функции поиск проводится только с начала поисковой переменной, и за одну операцию поиска можно найти только одну искомую величину, удовлетворяющую заданному отношению.

Для организации более сложных групповых операций поиска надо использовать оператор поиска данных в символьной переменной или символьном массиве. Оператор MAT SEARCH — оператор группового поиска данных, удовлетворяющих заданному Условию по всему содержимому поисковой переменной. Во время операции поиска просматривается вся поисковая переменная и находятся сразу все данные, удовлетворяющие заданному условию. Результатом операции поиска является список порядковых номеров начальных байтов найденных строк. Номера начальных байтов найденных строк символов записываются в список номеров, задаваемый символьной переменной. Общая форма оператора поиска имеет следующий вид:

MAT SEARCH	<поисковая переменная>,	<символ условия>,	<искомая величина>	TO	<список номеров>	[STEP <а.в.>]

где символ условия может быть задан следующими знаками:

```
< =     — меньше или равно;
<       — меньше;
=       — равно
>       — больше;
> =     — больше или равно;
<>      — не равно;
*       — равно по маске;
%       — равно с отождествлением;
#       — диапазон (в этом случае после символа условия в операторе записывается не одна, а две символьных
          переменных, содержимое которых определяет границы искомого диапазона).
```

Значение символьной константы или переменной, содержащей искомую величину, задается для поиска строк символов, удовлетворяющих заданному в операции условию. При поиске сравниваются

шестнадцатеричные коды искомой величины с содержимым поисковой переменной. Длина сравниваемых строк определяется длиной символьной переменной, содержащей искомую величину. При сравнении концевые пробелы искомой величины не входят в сравниваемое значение. Для того чтобы концевые пробелы искомой величины включались в число подлежащих сравнению символов, надо задавать искомую величину через функцию STR. В этом случае длина сравниваемых строк будет определяться длиной функции STR.

Номера начальных байт, найденных в поисковой переменной строк, последовательно записываются в символьную переменную, содержащую список номеров. Этот список создается в результате операции поиска. Номер в списке занимает два байта, в каждом из которых записывается значение номера в виде двоичного шестнадцатиразрядного числа. Поэтому минимальное количество байтов символьной переменной списка номеров должно быть равно двум.

Если параметр STEP (шаг просмотра) опущен, в операции сравнения последовательно участвуют строки поисковой переменной, начинающиеся с. первого, второго и т. д. байтов. Если задан параметр STEP, то в поисковом массиве последовательно сравниваются только строки, начинающиеся через каждые *k* байт, начиная с первого байта, где *k* — значение арифметического выражения. Абсолютное значение арифметического выражения должно быть в диапазоне:

$1 \le k \le 255.$

Если значение выражения положительно или параметр STEP опущен, поисковая переменная просматривается начиная с первого байта. Если значение выражения отрицательно, поисковая переменная просматривается, начиная с ее последнего байта. Если, например, задан STEP, равный —1, то при поиске последовательно просматриваются байты с конца поисковой переменной и до ее начала. При любом значении шага просмотра нумерация найденных строк символов идет от первого байта поисковой переменной.

Операция поиска заканчивается при выполнении одного из следующих условий:
проверены все возможные строки поисковой переменной;
символьная переменная списка номеров полностью заполнена номерами найденных строк.

Если операция поиска закончилась по первому условию, то в символьную переменную списка номеров в конец списка записываются два байта с содержимым HEX(0000). В частности если в операции поиска не было найдено ни одного искомого значения, первый и второй байты переменной списка номеров будут содержать двоичные нули.

Для иллюстрации выполнения операции поиска в символьной переменной по различным условиям рассмотрим следующий пример.

Размерность используемых ниже символьных переменных задана следующим образом:

```
10 DIM A⊙ 12, C⊙8
```

Поисковая переменная A⊙ содержит следующие данные:

```
20 A⊙ = «ГАБГАБВАБААБ»
```

Переменная C⊙ используется для хранения результата операции поиска (списка номеров найденных строк), а ее первоначальное содержимое задано следующим образом:

```
90 INIT(FF)C⊙
```

Ниже приведены шестнадцатеричные значения байтов переменной C⊙ после выполнения различных операций поиска:

```
:MAT SEARCH A⊙, = "A" TO C⊙: HEXPRINT C⊙
000200050008000A
:MAT SEARCH A⊙, = "АБ" TO C⊙: HEXPRINT C⊙
000200050008000B
:MAT SEARCH A⊙, = "АБ" TO C⊙ STEP 2:
HEXPRINT C⊙
0005000B0000FFFF
:MAT SEARCH A⊙, = "A" TO C⊙ STEP-1: HEXPRINT
C⊙
000B000A00080005
```

В последнем случае поиск буквы «А» ведется с конца переменной A⊙.

```
:MAT SEARCH STR (A⊙, 2), = "АБ" TO C⊙ STEP 3: HEXPRINT C⊙
0001000400070004
```

В предыдущем случае нумерация байтов идет с первого байта части переменной A⊙, выделяемой функцией STR(A⊙,2).

```
:MAT SEARCH A⊙, >"ГАБ"TO C⊙:HEXPRINT C⊙
00070000FFFFFFFF
```

Последний результат свидетельствует о том, что найденная последовательность символов — это «ВАБ». Дело в том, что код буквы «Г», равный HEX(E7) меньше кода буквы «В», равного коду HEX(F7).

```
:MAT SEARCH A⊙, < = "АБВ" TO C⊙: HEXPRINT C⊙
00020050008000A
:MAT SEARCH A⊙, <> "ГАБ" TO C⊙ STEP 3:
HEXPRINT C⊙
0007000A0000FFFF
:MAT SEARCH A⊙, = "Д" TO C⊙: HEXPRINT C⊙
0000FFFFFFFFFFFF
```

Переменная A⊙ не содержит кода буквы «Д», и поэтому результат операции поиска также не содержит ни одного номера найденного байта.

При задании в качестве условия поиска «равно по маске» каждый символ пробела в искомой величине считается совпадающим с любым символом в поисковой переменной. Поэтому операция поиска «равно по маске» используется при поиске некоторого контекста, в котором на определенных местах могут встретиться любые символы. Если в искомой величине нет символов пробела, операция поиска «равно по маске» эквивалентна операции поиска «равно». Рассмотрим результаты выполнения операции «равно по маске» для условий предыдущего примера:

```
:MAT SEARCH A⊙*"АВГА" TO C⊙: HEXPRINT C⊙
00020000FFFFFFFF
:MAT SEARCH A⊙*"АБ А" TO C⊙: HEXPRINT C⊙
000200050008 0000
```

В результате выполнения последней операции поиска найдены все искомые последовательность символов, в которых на третьем месте записан любой символ, т. е. «АБГА», «АБВА», «АБАА».

В условии «равно с отождествлением» так же, как и в условии «равно по маске», используется специальный символ, который считается совпадающим с последовательностью любых символов в поисковой переменной. В качестве такого символа в условии «равно с отождествлением» используется код HEX(00). При выполнении операции поиска каждый код HEX(00) считается совпадающим с последовательностью символов любой длины, и в том числе с пустой последовательностью, не содержащей ни одного символа, т. е. код HEX (00) в искомой величине отождествляется с любой строкой символов в поисковой переменной. Результаты операции поиска по этому условию записываются в отличие от рассмотренных условий поиска в четыре последовательных байта списка номеров. Первые два байта так же, как и в других случаях, содержат двоичное значение номера первого байта найденной последовательности. Последние два байта содержат двоичное значение длины в байтах найденной последовательности.

Дополним задание переменных предыдущего примера:

```
40 DIM K⊙3, E⊙20 : INIT(FF) E⊙
```

Переменная E⊙ используется для хранения результатов поиска, а переменная K⊙ будет содержать искомую последовательность. В результате выполнения операции поиска «равно с отождествлением» получим следующие результаты:

```
:K⊙ = "АА" : STR (K⊙,3,1) = HEX(00)
:MAT SEARCH A⊙, %K⊙ TO E⊙ : HEXPRINT E⊙
0002000400050004000800030004000A00020000FFFF
```

В результате операции поиска будут найдены последовательности: «АБГА», «АБВА», «АБА» и «АА».

Общая форма оператора поиска при задании условия диапазоном имеет следующий вид:

```
MAT SEARCH  <поисковая переменная>,  <нижняя граница>,  <верхняя граница>  TO  <список номеров>  [STEP <а.в.>]
```

где содержимое символьных переменных нижней и верхней границ диапазона определяет поиск цепочек символов X, удовлетворяющих условию:

<нижняя граница > ≤ X ≤ <верхняя граница>

Сравниваются шестнадцатеричные коды символов поисковой переменной с шестнадцатеричными кодами строк нижней и верхней границ диапазона. Длина сравниваемых строк символов определяется длиной строки нижней границы при сравнении «меньше или равно» и длиной строки верхней границы при сравнении «больше или равно». В общем случае эти длины могут быть различными.

Пусть для примера в массиве C⊙ () хранятся в символьном виде двухзначные числа. Необходимо найти элементы массива, содержащие числа в заданном диапазоне, например в диапазоне (25,36). Содержимое поисковой переменной и переменной списка номеров определено и задано следующим образом:

```
10  DIM C⊙(12)2,  E⊙12  :  INIT(FF)  E⊙
20  STR (C⊙( ), 1)=  "13012850993025241036"
```

Выполнение операции поиска и печать результатов содержимого списка, найденных номеров байтов, в шестнадцатеричном виде задаются следующими операторами:

```
30 MAT  SEARCH  C⊙( ),#"25",  "36"TO  E⊙  STEP2
40  HEXPRINT  E⊙
```

Для вывода результатов операции поиска в виде номеров элементов массива, содержимое которых находится в заданном диапазоне, выполним преобразование результатов.

```
50  I = 1
60  IF STR (E⊙, I, 2) = HEX (0000) THEN 90
70  PRINT "ЭЛЕМЕНТ"; (VAL (STR (E⊙, I, 2), 2) + l)/2
80  I=I+2: GOTO 60
90  STOP
```

После запуска программы получим

```
:RUN
0005000B000D00130000FFFF
ЭЛЕМЕНТ  3
ЭЛЕМЕНТ  6
ЭЛЕМЕНТ  7
ЭЛЕМЕНТ  10
:__
```

Пример 15.4. В элементах массива B⊙ () записаны наименования материалов, а в соответствующих элементах числового массива C() содержится стоимость этих материалов. Следующая программа печатает стоимость вводимого материала.

```
10  DIM B⊙ (10, 30),C (10, 30),K⊙2
20  LINPUT "МАТЕРИАЛ", A⊙
30  MAT SEARCH B⊙ ( ). = A⊙ TO K⊙ STEP 16
40  IF K⊙<> HEX(0000) THEN 60
50  PRINT "ТАКОГО МАТЕРИАЛА НЕТ" : GOTO 20
60  M% = VAL (K⊙, 2)/16
70  I%=M% / 30+1
80  J%=M%- (I%-1)*30+1
90  PRINT "СТОИМОСТЬ ";A⊙; C(I%, J%)
```

Размер элемента двумерного массива B⊙ () задан в строке 0 программы по умолчанию равным 16. Поэтому в строке 30 программы в операторе поиска строки массива B⊙ () просматриваются через 16 байт. В строках 60—80 программы найденное значение первого номера байта элемента, содержащего искомый материал, пересчитывается в номера строк и столбцов найденного элемента.

15.2. Копирование содержимого символьных переменных

Операции копирования (переписи) содержимого символьных переменных и массивов выполняются оператором MAT COPY. Оператор MAT COPY переписывает данные из входной символьной переменной или ее части в выходную символьную переменную или ее часть. Данные переписываются

последовательно по байтам. При переписи границы элементов массива игнорируются, а содержимое массива или его части рассматривается как одна строка символов. Общая форма оператора переписи имеет следующий вид:

```
MAT  COPY[—]< обозначение входной      TO [—]< обозначение выходной
             символьной                        символьной переменной>
             переменной>
```

где необязательный знак «минус» перед обозначением входной или выходной переменной указывает на то, что данные извлекаются из входной переменной или соответственно переписываются в выходную переменную в обратном порядке. Если знак «минус» отсутствует, данные извлекаются или переписываются в прямом порядке. Операция переписи данных заканчивается, когда заполняется вся выходная переменная или ее часть. Если переписываемых байтов недостаточно для заполнения всей переменной или ее части, в оставшиеся байты выходной переменной записываются символы пробела. При отсутствии знака «минус» перед обозначением входной переменной переписываемые байты извлекаются последовательно в прямом порядке, начиная с первого байта. Если знак «минус» стоит перед обозначением входной переменной, переписываемые байты извлекаются в обратном порядке, начиная с последнего байта входной переменной. При отсутствии знака «минус» перед обозначением выходной переменной переписываемые байты записываются в выходную переменную в прямом порядке, начиная с ее первого байта. Если знак «минус» стоит перед обозначением выходной переменной, переписываемые байты записываются в выходную переменную в обратном порядке, начиная с последнего байта выходной переменной. В операции копирования в качестве входной и выходной переменных может использоваться одна и та же переменная или ее часть. Для выделения части символьной переменной или символьного массива используется функция STR.

Рассмотрим результаты выполнения операции копирования при задании различных условий переписи данных. Пусть в качестве входной символьной переменной используется переменная А⊙, а в качестве выходной переменной — В⊙. Размерность этих переменных задана следующим образом:

```
10  DIM   A⊙5, B⊙7
```

В переменную А⊙ записывается содержимое, определенное следующей операцией присваивания:

```
20  A⊙ = "АБВГД"
```

Далее приведены результаты выполнения соответствующих операций переписи данных из переменной А⊙ в переменную В⊙. Данные извлекаются и записываются как в прямом, так и в обратном порядках. Содержимое переменной В⊙ после переписи для иллюстрации полученных результатов распечатывается как в символьном виде, так и шестнадцатеричными значениями

```
:MAT COPY A⊙ TO B⊙: PRINT B⊙: HEXPRINT B⊙
АБВГД
C1C2D7C7C42020
:MAT COPY - A⊙ TO B⊙: PRINT B⊙: HEXPRINT B⊙
ДГВБА
C4C7D7C2C12020
:MAT COPY A⊙TO -B⊙: PRINT B⊙: HEXPRINT B⊙
   ДГВБА
2020C4C7D7C2C1
:MAT COPY -A⊙ TO -B⊙: PRINT B⊙: HEXPRINT B⊙
   АБВГД
2020C1C2D7C7C4
```

Пример 15.5. В программе редактируется содержимое символьного массива С⊙( ). При операции редактирования необходимо вставлять в переменную С⊙( ) некоторое количество символов или удалять из переменной С⊙( ) некоторое количество символов. Ниже приведен фрагмент программы, содержащей подпрограммы вставки и удаления строк символов из символьного буфера.

```
10   DIM C⊙ (20, 80)1, A⊙100
. . .
500   REM ПРИМЕР ВСТАВКИ   5 СИМВОЛОВ С 251 БАЙТА
510   GOSUB'100 (251, "СЛОВО")
. . .
600   REM ПРИМЕР УДАЛЕНИЯ 10 СИМВОЛОВ С 300 БАЙТА
```

```
610  GOSUB'101 (300, 10)
. . .
1000  REM ПОДПРОГРАММА ВСТАВКИ СИМВОЛОВ В ПОЗИЦИЮ
K
1010  DEFFN100 (K, A⊙): L=LEN (A⊙)
1020  MAT COPY – STR (C⊙( ), K, 1601-K-L) TO
- STR (C⊙( ), K+L)
1030  STR (C⊙ ( ), K, L) = A⊙: RETURN
1040  REM ПОДПРОГРАММА УДАЛЕНИЯ СИМВОЛОВ ИЗ
ПОЗИЦИИ K
1050  DEFFN'101 (K, L)
1060  MAT COPY STR (C⊙( ), K+L) TO STR (C⊙, K)
1070  RETURN
```

15.3. Замена и перекодировка содержимого символьных массивов

Замена в содержимом символьной переменной одного контекста на другой осуществляется с помощью оператора REPLACE. Этот оператор позволяет находить и заменять определенные цепочки символов на другие цепочки символов в содержимом символьной переменной и подсчитывать количество совершенных замен. При замене в содержимом символьного массива, так же как и в операторах поиска и копирования, массив рассматривается как одна строка символов без границ между элементами. Общая форма оператора замены имеет следующий вид:

```
REPLACE   <числовая переменная>,   <символьная переменная>,   <искомая строка>   [,<заменяющая строка>]
```

где числовая переменная предназначена для записи в нее числа совершенных в содержимом символьной переменной замен посла выполнения оператора. Если не произойдет ни одной замены, значение этой переменной будет равно нулю. Символьная переменная обозначает изменяемое содержимое. До операции замены в переменной хранится исходный текст, а после операции замены в нее записывается новый текст. Искомая строка может быть символьной константой или переменной, содержащей последовательность заменяемых символов. Заменяющая строка может быть символьной константой или переменной, содержащей последовательность символов замены. Этот параметр оператора замены необязателен. Если его нет, то искомая строка удаляется из содержимого символьной переменной.

При выполнении операции замены длины искомой и заменяющей строк могут быть различными. Поэтому в результате замены содержимое символьной переменной без учета концевых пробелов может быть больше или меньше исходного содержимого. В последнем случае содержимое переменной дополняется необходимым количеством символов пробела.

Пусть, например, требуется заменить во всем содержимом символьной переменной A⊙ ( ) фамилию «ИВАНОВ» на фамилию и инициалы «ИВАНОВ В. И.». Для требуемой замены достаточно выполнить оператор

```
: REPLACE   K, A⊙ ( ),"ИВАНОВ",   "ИВАНОВ В. И."
```

После выполнения оператора в переменной К будет записано число найденных и замененных фамилий «ИВАНОВ».

Оператор замены можно использовать и просто для подсчета встречаемых в тексте последовательностей символов. Для этого Достаточно использовать одно и то же значение искомой и заменяющей цепочки символов. Например, необходимо выяснить, сколько раз в содержимом переменной C⊙ ( ) встречается слово «ШИФР». Для подсчета количества искомых строк надо выполнить оператор

```
: REPLACE  A, C⊙ ( ) ,"ШИФР", "ШИФР"
```

После выполнения оператора искомое количество будет записано в переменную А.

При необходимости удаления из содержимого символьной переменной некоторого контекста оператор замены используется без последнего параметра. Пусть, например, из содержимого переменной B⊙ ( ) требуется удалить все пробелы между символами, кроме концевых пробелов. Для этого выполним оператор

```
:REPLACE   M,   B⊙ ( ),   HEX (20)
```

После выполнения оператора в содержимом переменной B⊙ ( ) останутся только концевые пробелы.

П р и м е р 15.6. В программе в массиве E⊙( ) хранится введенный с клавиатуры текст, при вводе которого между словами может оказаться более одного символа-разделителя — пробела. Следующий фрагмент программы преобразует содержимое исходного текста, оставляя только по одному пробелу между словами:

```
. . .
1000  REPLACE  K, E⊙ ( ), HEX(2020), HEX(20)
1010  IF  K<>0  THEN  1000
. . .
```

Замена двух подряд идущих пробелов осуществляется оператором в строке 1000. Операция производится до тех пор, пока в исходном тексте не останется только по одному пробелу между словами. Оператор в строке 1000 повторяется несколько раз, так как в исходном тексте может встретиться несколько подряд идущих пробелов.

П р и м е р 15.7. При редактировании некоторого символьного значения требуется выполнять операцию замены текстов, вводимых с клавиатуры, а также исключать вводимые с клавиатуры тексты. Следующая часть программы осуществляет такое изменение текста:

```
10  DIM  B⊙  (1000), A⊙80, C⊙80
. . .
1000  REM ПОДПРОГРАММА ЗАМЕНЫ ТЕКСТА
1010  PRINT AT (1, 1, 320) : PRINT "ЧТО ЗАМЕНИТЬ?"
1020  PRINT AT (2, 1): LINPUT A⊙
1030  PRINT AT(3, 1): PRINT "НА ЧТО ЗАМЕНИТЬ?"
1040  C⊙ A⊙: PRINT AT (4, 1): LINPUT  C⊙
1050  REPLACE K, B⊙( ), A⊙, C⊙
1060  RETURN
. . .
1100  REM ПОДПРОГРАММА УДАЛЕНИЯ ТЕКСТА
1110  PRINT AT (1, 1, 160): PRINT "ЧТО УДАЛИТЬ?"
1120  PRINT AT (2, 1): LINPUT  A⊙
1130  REPLACE K, B⊙( ), A⊙
1140  RETURN
```

Оператор ⊙TRAN позволяет производить быструю перекодировку всего содержимого символьной переменной в соответствии с задаваемой таблицей символов. Этот оператор производит групповую операцию замены одних символов на другие. Общая форма оператора имеет следующий вид:

⊙TRAN (<символьная переменная>, <таблица перекодировки>) [hh][R]

где символьная переменная содержит исходные данные, подлежащие перекодировке. Таблица перекодировки, задаваемая символьной переменной, содержит список кодов, в соответствии с которыми происходит операция. Две шестнадцатеричные цифры hh определяют маску, используемую при преобразовании данных. Маска в свою очередь определяет двоичные разряды в каждом символе исходных данных, которые заменяются двоичными нулями, если соответствующие разряды маски равны нулю. Маска может не использоваться. Параметр R определяет способ задания таблицы перекодировки. Если он указан, таблица перекодировки задается списком заменяемых кодов. Без параметра R таблица перекодировки должна содержать кодовую таблицу. Таким образом, можно использовать две формы оператора ⊙TRAN в зависимости от требуемого способа перекодировки.

В списковой форме оператора ⊙TRAN с параметром R второй аргумент оператора обозначает список кодов. Этот список состоит из нескольких пар байтов. В каждой паре байтов второй байт содержит код заменяемого во всей переменной символа. Содержимое первого байта каждой пары определяет код символа, на который заменяются перекодируемые символы. Содержимое последних двух байтов списка должно быть равно коду HEX(20), т. е. двум символам пробела.

Перекодировка в списковой форме оператора ⊙TRAN предполагает следующую последовательность действий:

1) извлекается исходное содержимое очередного байта символьной переменной, начиная с ее первого байта. Это содержимое изменяется в соответствии с маской, если она задана;

2) полученный код последовательно сравнивается со всеми кодами четных байтов списка. Как только данный код совпадает с содержимым какого-либо четного байта, сравнение заканчивается, и

исходный байт символьной переменной заменяется на нечетный байт из пары байтов списка;

3) если совпадения не произойдет, исходный байт символьной переменной заменяется на результат операции 1. При этом содержимое этого байта не изменяется, если маска не была задана, или изменится в соответствии с маской;

4) далее повторяется операция 1, пока не будут извлечены все байты аргумента, и на этом операция перекодировки заканчивается.

Пример 15.8. В массиве A⊙ ( ) необходимо заменить все символы * на символы ?. Остальные символы не должны меняться.

```
10  DIM  A⊙ (10), C⊙4
20  C⊙ = "?*"
30  ⊙TRAN  (A⊙( ), C⊙) R
```

Пример 15.9. В переменной A⊙ все коды HEX (0A) (новая строка) надо заменить на коды HEX(85) (возврат каретки).

```
10  DIM B⊙4
20  B⊙ = HEX (850A)
30  ⊙TRAN (A⊙, B⊙) R
```

В табличной форме оператора ⊙TRAN без параметра R второй аргумент обозначает таблицу перекодировки. Размер таблицы определяется размером символьного аргумента. Каждый байт таблицы содержит символ перекодировки. Так как количество различных кодов равно 256, максимальный размер таблицы перекодировки составляет 256 байт.

Перекодировка в табличной форме оператора ⊙TRAN происходит следующим образом:

1) извлекается исходное содержимое очередного байта символьной переменной, начиная с ее первого байта. Это содержимое изменяется в соответствии с маской, если она задана;

2) полученный код преобразуется в число, к которому прибавляется единица. Этот результат определяет номер байта в таблице перекодировки. Содержимое этого байта заменяет исходный байт в символьной переменной. Если, например, содержимое очередного исходного байта символьной переменной содержит код символа 1, т. е. HEX(31), то этот код преобразуется в десятичное число 49, равное 3·16+1. К числу 49 прибавляется единица, и содержимое 50-го байта таблицы заменяет содержимое исходного байта символьной переменной;

3) если в таблице содержится меньше байтов, чем полученный при выполнении операции 2 номер, то соответствующий байт переменной заменяется на результат операции 1, т. е. либо содержимое этого байта не изменится, если маска не была задана, либо изменится в соответствии с маской;

4) операция 1 повторяется, пока не будут извлечены все байты символьной переменной, и на этом перекодировка заканчивается.

Пример 15.10. Программа распечатывает шестнадцатеричные значения младшей половины байта вводимого символа. Так как эти значения лежат в диапазоне от 0 до F, то длина используемой таблицы перекодировки равна 16 байтам. Для извлечения только четырех младших разрядов исходных символов в операторе ⊙TRAN используется маска со значением HEX (0F). Поэтому во время выполнения оператора разряды с 7-го по 4-й исходного байта заменятся нулями и далее перекодируются только младшие разряды байта.

```
10  DIM A⊙1
20  T⊙="0123456789ABCDEF"
30  INPUT A⊙
40  HEXPR1NT A⊙
50  ⊙TRAN (A⊙, T⊙) 0F
60  PRINT A⊙
```

Ниже приведены примеры работы данной программы. Для удобства сравнения программа сначала печатает шестнадцатеричные значения вводимых символов.

```
:RUN
?*
2A
A
:RUN
? 9
```

```
39
9
:RUN
? A
41
1
:RUN
? HEX (0F)
0F
F
```

Пример 5.11. В символьной переменной А⊙, предназначенной для вывода, могут встретиться управляющие коды, т. е. коды с шестнадцатеричными значениями меньшими кода пробела — HEX (20). При выводе в содержимом символьной переменной они могут быть восприняты как служебные команды управления устройством вывода. Поэтому при попытке вывода содержимого символьной переменной на экран или печатающее устройство текст может быть распечатан неверно. Избежать подобной ситуации можно, предварительно заменив в символьной переменной все коды, меньшие символа пробела, на печатные символы. Использование оператора перекодировки позволяет осуществить подобную замену. Размер таблицы перекодировки определяется исходя из количества кодов, меньших кода пробела. Количество таких кодов начиная с кода HEX(00) до кода HEX (1F) равно 32. В данном примере все непечатаемые коды заменятся на символ точки.

```
10 DIM  C⊙32, A⊙25:  INIT(".")  C⊙
. . .
500  ⊙TRAN (A⊙, C⊙)
. . .
```

В заключение отметим, что оператор табличной перекодировки удобно использовать при обмене данными между микроЭВМ «Искра 226» и другими ЭВМ, например ЕС ЭВМ. В этом случае таблицы размером по 256 байт должны задавать соответствие между кодом КОИ-8 (код микроЭВМ «Искра 226») и кодом ДКОИ (код обмена данными ЕС ЭВМ). Применение оператора табличной перекодировки будет необходимо и перед выполнением сортировки текстов на русском языке, поскольку в коде КОИ-8 символы русских букв не расположены в алфавитном порядке. Таблица перекодировки в этом случае должна быть составлена таким образом, чтобы новые коды букв возрастали в порядке следования букв в русском алфавите.

Глава 16. УПРАВЛЕНИЕ ФОРМАТОМ ПЕЧАТИ

16.1. Операторы, задающие формат печати

Оператор PRINTUSING и оператор задания формата % используются совместно для управления размещением данных при печати на экране дисплея или АЦПУ. Чтобы использовать эти операторы, записывается оператор %, в котором указывается нужный формат печати. Затем, когда нужно напечатать какое-либо значение в соответствии с этим форматом, используется оператор PRINTUSING. В нем содержится ссылка на номер строки, в которой находится оператор с описанием формата печати. Оператор % является неисполняемым и может размещаться в любой строке программы. Для того чтобы сравнить операторы PRINT и PRINTUSING, рассмотрим следующий пример.

Пример 16.1

```
100   PRINT "ОПЕРАТОР PRINT", "ОПЕРАТОР
PRINTUSING"
110   FOR K=1 TO 8
120   READ A
130   PRINT A, ,
140   PRINTUSING 160, A
150   NEXT K
160   %   # # # # #  .# #
170   DATA 23400, 5, 7.3, 6, 13.06, 140.767685341,
0,    .01
```

В результате выполнения работы этой программы на экране будет напечатано:

Оператор PRINT	Оператор PRINTUSING
23400	23400.00
5	5.00
7.3	7.30
.6	0.60
13.06	13.06
140.767685341	140.76
0	0.00
0.01	0.01

В этом примере читаются числа из списка оператора DATA и печатаются сначала с помощью оператора PRINT, затем — оператора PRINTUSING. Формат печати чисел, вводимых по оператору PRINT, определяется их значением. В отличие от этого способа печати вывод чисел по оператору PRINTUSING осуществляется по формату, заданному пользователем. В строке 140

```
140 PRINTUSING 160, A
```

160 означает номер строки, где записан оператор задания формата печати. В программе этот оператор записывается с помощью символа %. Переменная A в строке 140 является элементом печати. Каждый раз, когда выполняется строка 140, оператор PRINTUSING обеспечивает печать значения переменной A по формату, заданному в строке 160. Этот формат записан в виде # # # # # # # #. В этом описании формата символ # используется для обозначения разряда числа, подлежащего печати. Точка указывает положение десятичной точки и воспроизводится при печати числа. Два символа # # справа от точки показывают, что должны печататься только две цифры дробной части числа. Таким образом, если печатается столбец чисел, то печать выравнивается по десятичной точке.

Один оператор % может содержать несколько описаний формата. Следующий пример иллюстрирует случай использования оператора PRINTUSING с четырьмя элементами печати, ссылающийся на оператор %, который содержит четыре описания формата печати.

Пример 16.2

```
100 PRINTUSING 110, 145.76, 145.76, 145.76, 145.76
110%### ###.# ###.## ###.### ###.###
```

В результате выполнения этой программы будет напечатано

```
145  145.7  145.76  145.760
```

При выполнении оператора PRINTUSING в строке 100 машина выбирает первый элемент печати (145.76) и начинает подставлять цифры этого числа вместо знаков # в первом описании формата.

Цифра 1 подставляется вместо первого символа #, цифра 4 — вместо второго, цифра 5— вместо третьего. Так как система не находит больше знаков # в первом описании формата, то остальные цифры первого элемента печати не будут учитываться. Система перейдет ко второму элементу печати и начнет подставлять его цифры на место знаков # во втором описании формата в операторе %. После подстановки цифры 7 описание формата второго элемента закончится и цифра 6 не будет напечатана. Аналогичная процедура будет выполнена с третьим и четвертым элементами печати. В случае четвертого элемента после подстановки цифр 1, 4, 5, 7 и 6 в описании формата один знак # не будет заполнен, поэтому на его место автоматически будет подставлена цифра 0. После того как список элементов печати в операторе PRINTUSING будет исчерпан, в машине создается образ печатной строки, аналогичный описанному в строке 110, но с замещенными на конкретные цифры знаками #. Эта строка будет выводиться на экран дисплея, начиная с позиции, где находится в данный момент курсор. Пробелы в описании формата между группами символов будут воспроизводиться при печати.

В последнем формате в строке 110 есть лишний символ справа. Этот символ замещается нулем при выполнении оператора PRINTUSING. Если же обратиться к примеру 16.1, то видно, что, за исключением первого числа, в задании формата печати для всех чисел есть символы # лишние слева. Эти символы замещаются не нулями, а пробелами, т. е. при печати первого числа (23400) выполняется подстановка

```
# # # # #  .# #   — формат печати
2 3 4 5 0  .0 0   — печатаемый элемент
```

При печати второго числа (5) выполняется подстановка пробелов для символов #, стоящих слева от первой цифры

```
#####  .##   — формат печати
     5  .00   — печатаемый элемент
```

При печати чисел, являющихся дробями, вместо первой цифры целой части чисел подставляется нуль.

Чтобы обеспечить вывод данных с помощью оператора PRIN TUSING на АЦПУ, необходимо включить в программу до употребления оператора PRINTUSING оператор SELECT PRINT ОС либо использовать возможность непосредственного задания устройства печати в операторе PRINTUSING.

```
PRINTUSING/0C.  160,  A
```

Элементом печати оператора PRINTUSING могут быть числовая или символьная константа и переменная, а также выражение. Элементы печати оператора PRINTUSING разделяются запятыми или точкой с запятой. Запятая является простым разделителем элементов печати (Случай употребления «;» рассматривается в следующем разделе.)

16.2. Задание формата

Рассмотрим подробнее различные возможности задания формата печати с помощью оператора задания формата %.

Допускается включать в оператор % любые символы, кроме символов #, запятой и точки с запятой. Часто такое добавление поясняющего текста делает выходную печать более понятной. Рассмотрим пример простой программы, обеспечивающей расчет процента выполнения плана.

Пример 16.3

```
100  REM ПРОГРАММА РАСЧЕТА ПРОЦЕНТА
110  INPUT "ВВЕДИТЕ ПЛАН И ОТЧЕТ", P, R
120  REM ПЕЧАТЬ РЕЗУЛЬТАТА
130  PRINTUSING  140,  P,  R,  R*100/P
140  %ПЛАН,### .# ОТЧЕТ ### .# ПРОЦЕНТ
ВЫПОЛНЕНИЯ= ###  .##
```

В результате выполнения этой подпрограммы после ввода значений плана и отчета (строка 110)

ВВЕДИТЕ ПЛАН И ОТЧЕТ? 45, 54

будет напечатано

```
ПЛАН = 45.0    ОТЧЕТ = 54.0    ПРОЦЕНТ ВЫПОЛНЕНИЯ= 120.00
```

В строке 130 в операторе PRINTUSING три элемента печати, причем третий элемент задан в виде выражения R*100/P. Это выражение вычисляется и результат подставляется цифра за цифрою в соответствующее ему описание формата в операторе % . Пояснительный текст, включенный в оператор % (слова «ПЛАН = », «ОТЧЕТ = », «ПРОЦЕНТ ВЫПОЛНЕНИЯ = », а также пробелы, стоящие между символом % и словом «ПЛАН», воспроизводятся при печати.

Допускается использование оператора %, в котором нет символов описания формата, а содержится лишь символьная константа. Таким способом задания оператора % можно, например, воспользоваться для печати таблицы, размещение элементов заголовка которой будет соответствовать расположению столбцов.

Пример 16.4

```
100  %   ПЛАН    ОТЧЕТ
110  %   ####    ####
 . . . . . .
200  PRINTUSING 100
210  PRINTUSING ll0, P, R
220  PRINTUSING 110, 20, 25
```

В этом примере оператор % в строке 100 состоит лишь из символов, отличных от символа #. В строке 200 системы дается Указание вывести на печать содержимое оператора % из строки 100. Заметим, что в строке 200 в операторе PRINTUSING нет списка элементов печати. В результате работы программы будет Напечатано

```
ПЛАН          ОТЧЕТ
45              54
20              25
```

Расположение столбцов чисел точно соответствует заголовку. В предыдущих примерах на печать выводились лишь положительные числа. Для того чтобы обеспечить вывод чисел со знаком, необходимо перед описанием формата указать знак — или +. Если перед форматом печати в операторе % стоит знак —, то отрицательные числа печатаются по этому формату со знаком «минус», а положительные числа — с символом пробела перед числом. Если перед описанием формата стоит знак +, перед числом печатается символ + или — в зависимости от знака числа. Приведем пример использования знаков «плюс» и «минус» в описании формата.

Пример 16.5

```
100  RINTUSING          130,  15.35,  - 6.27
110  RINTUSING          140,  15.35,  - 6.27
120  RINTUSING          150,  15.35,  - 6.27
130       ###.##            ###.##
140     - ###.##          - ###.##
150     + ##  #.##        + ###.##
```

В результате выполнения программы будет напечатано

```
    15.35          6.27
    15.35        – 6.27
  + 15.35        – 6.27
```

Первая строка печатается по формату, описанному в строи 130. Поскольку он не содержит знаков + или — перед форматом, то числа печатаются без знака. Это заметно при печати второго числа: в этом случае «теряется» знак «минус». При печати по формату в строке 140 перед описанием форматов печати стоит —, поэтому числа печатаются со знаком, но вместо знака + у первого числа печатается пробел. В третьей строке знак числа воспроизводится как в случае отрицательного, так I положительного числа. Ниже приводится еще один пример использования формата печати со знаком.

Пример 16.6

```
100  PRINTUSING 120
110  PRINTUSING 130, P, R, R-P
120  %  ПЛАН     ОТЧЕТ    , ± К ПЛАНУ
130  %   ###        ###.       +###
```

Операторы PRINTUSING и % не обеспечивают округления чисел. Десятичные цифры, для которых в описании формата хватило знаков #, просто отбрасываются. Например, при печати переменной P, значение которой равно 71.378, по формату ###.# будет напечатано 71.3. Чтобы напечатать это значение, округленное до десятых, необходимо использовать функцию ROUND, например

```
PRINTUSING  130,  ROUND(P, l)
```

Итак, если в описании формата справа от десятичной точки указано число знаков меньше количества дробных цифр в числе, оставшиеся дробные цифры не печатаются. Иная ситуация возникает в том случае, когда для изображения цифр целой части числа знаков # недостаточно.

Пример 16.7. Если попытаться напечатать число 5555 по формату # # #

```
100  PRINTUSING  110,  5555,  375,  147.52
100  %  ###       ###     ###
```

то вместо числа 5555 будет напечатано описание формата, т. е.

```
###    375    147
```

Появление знаков # в выходной печати говорит о том, что для правильной печати чисел нужно отредактировать формат, добавив дополнительные символы # для целой части соответствующих чисел.

С помощью оператора PRINTUSING числа могут выводиться в экспоненциальной форме. Для этого в описание формата включаются четыре символа /\ после описания формата мантиссы числа.

Пример 16.8

```
100  A=6.57
110  PRINTUSING  120A, 2.3E—9
120  %  #.#/\/\/\  ##/\/\/\
```

В результате выполнения программы будет напечатано

```
6.5E+00        23E—10
```

Оператор PRINTUSING можно использовать также для вывода на печать символьных данных в виде символьных констант или переменных. В этом случае описание формата в операторе % для таких элементов печати состоит только из символов #. Если длина символьного элемента больше числа символов #, остальные символы не печатаются.

Пример 16.9

```
100  A⊙, "ПР0ЕКТ ПЛАНА"
110  PRINTUSING 130, A⊙
120  PRINTUSING 140, A⊙
130  %  ##########
140  %   ########
```

В результате выполнения программы будет напечатано

ПРОЕКТ ПЛАНА
ПРОЕКТ П

Список элементов печати оператора PRINTUSING может состоять из числовых и символьных элементов. Следующий пример демонстрирует случай печати строк, состоящих из символьных 9 числовых элементов.

Пример 16.10. Требуется напечатать отчет о выполнении плана производства по 10 предприятиям. Пусть наименование предприятий хранятся в массиве A⊙ (), значения планов — в массиве P(), отчетов — в массиве R( ).

```
100  DIM A⊙ (10) 20, P (10), R (10)
110  REM ВВОД ЗНАЧЕНИЙ ЭЛЕМЕНТОВ МАССИВОВ A⊙( ),
P( ), P( )
120  MAT INPUT A⊙( ), P( ), R( )
130  PRINTUSING 170
140  FOR K = 1 TO 10
150  PRINTUSING 180, A⊙(K), P(K), P(K), R(K)-P(K)
160  NEXT K
170  %  ПРЕДПРИЯТИЕ                    ПЛАН     ОТЧЕТ     ±  К
ПЛАНУ
180  %  ####################    ###.#    ###.#
.+###.#
```

В результате выполнения программы будет напечатана таблица, содержащая четыре графы в соответствии с заголовком, описанным в строке 130.

Если число элементов печати в операторе PRINTUSING больше числа описаний формата в операторе %, машина автоматически осуществляет перевод строки, как только описание формата полностью использовано. После этого оператор % используется повторно для печати остальных элементов оператора PRINTUSING. В следующем примере оператор % используется три раза одним оператором PRINTUSING.

Пример 16.11

```
100  DIM  A(3),  B(3)
110  % НОМЕР  СКЛАДА    КОЛИЧЕСТВО
120  %       ###                 ####.#
130  REM ВВОД ЗНАЧЕНИЙ  ЭЛЕМЕНТОВ МАССИВОВ A( )  и
B( )
140  MAT  INPUT  A( ),  B( )
150  PRINTUSING  110
160  PRINTUSING  120,  A(1),  B(1),  A(2),  B(2),  A(3),
B(3)
```

В результате выполнения предыдущей программы будет напечатано

НОМЕР СКЛАДА	КОЛИЧЕСТВО
13	1320.2
21	725.7
32	5526,5

(цифры взяты условно)

В этом примере в строке 160 выводятся на печать три строки чисел посредством тройного использования оператора % в строке 120. При выполнении строки 160 сначала печатается значение A(1)

по формату # # #, затем — значение B(1) по формату # # #.#. Список элементов не исчерпан, но в операторе % нет больше описаний формата. Происходит переход на начало следующей строки. Затем оператор % используется для вывода A(2) и B(2) и т. д.

При многократном использовании описания формата можно подавить перевод курсора на новую строку посредством использования точки с запятой при записи списка элементов печати в операторе PRINTUSING.

Пример 16.12

```
100  INPUT  A,   B,   P,   R
105  PRINTUSING 120: PRINTUSING 130
110  PRINTUSING 140, A, B; P, R
120  %  ПЛАН    ОТЧЕТ   ПЛАН    ОТЧЕТ
130  %                        С НАЧАЛА ГОДА
140  %      ###     ###
```

В результате выполнения программы будет напечатано

```
ПЛАН         ОТЧЕТ          ПЛАН         ОТЧЕТ
                                С  НАЧАЛА  ГОДА
  150            156             372           381
```

Формат печати в строке 140 содержит два описания, а список оператора PRINTUSING — четыре элемента печати. Точка с запятой после переменной B в строке 110 подавляет перевод курсора на новую строку. Печать переменных P и R осуществляется по формату строки 140, начиная со следующей позиции после печати переменной B.

Пример 16.13. Требуется напечатать значения массива A () в одну строку, при этом каждый элемент A() следует печатать по формату # # # .#.

```
100  DIM A(6)
110  REM ВВОД ЭЛЕМЕНТОВ МАССИВА A( )
120  MAT INPUT A
200  F OR K=l TO 6
210  PRINTUSING 250, A(K);
220  NEXT K
230  PRINT
240  PRINT "КОНЕЦ СТРОКИ"
250  %# # # .# #
```

В результате выполнения программы будет напечатано

```
256.12    140.15    105.30    95.31    551.01    6.15
КОНЕЦ  СТРОКИ
```

В этом примере формат печати в строке 250 используется оператором PRINTUSING в строке 210 шесть раз. Так как в операторе после A (K) стоит точка с запятой, после печати значения A (K) перевода курсора на новую строку не происходит, значения массива A() печатаются по заданному формату один за другим на одной строке. Заметим, что формат в строке 250 содержит символ пробела справа от последнего символа #. Этот пробел воспроизводится при печати и благодаря ему создается интервал между значениями массива A(). Чтобы убедиться в его существовании, можно вызвать строку 250 на редактирование

```
*250   %    # # # .# #_
```

Оператор PRINT в строке 230 обеспечивает вывод последующих элементов печати с новой строки. Если его удалить из программы, то текст «КОНЕЦ СТРОКИ» будет напечатан вслед за последним числом.

Помимо описанных возможностей формат печати может задаваться также непосредственно в операторе PRINTUSING в виде символьной константы или символьной переменной, содержащей символы описания формата # и другие символы, например

```
100  PRINTUSING  "# # #    # # .#",   A,   B
110  A⊙="ИТОГО= # # #       ПРИБЫЛЬ = # # # .#"
120  PRINTUSING  A⊙,   P,   R-P
130  PRINTUSING   "# .# # /\\\\" ,  C
```

П р и м е р 16.14. Требуется написать программу для печати формы

```
НАИМЕНОВАНИЕ        1982 Г,        1983 Г,
ПОКАЗАТЕЛЬ 1

 . . . . . .
ПОКАЗАТЕЛЬ 10
```

При этом значения показателей нужно печатать по различным не заданным заранее форматам. Пусть наименования показателей хранятся в массиве С⊙ (), значения показателей за **1982 г.** — в массиве А(), за 1983 г. — в массиве В(). Прежде чем печатать форму, введем массив форматов Х⊙ (), в соответствии с которыми будем печатать строки формы.

```
10  DIM C⊙(10)20, A(10), B(10)
20  DIM X⊙(10)10: REM МАССИВ ФОРМАТОВ
30  FOR K=1 TO 10
40  INPUT "ВВЕДИТЕ НАИМЕНОВАНИЕ", C⊙(K)
50  INPUT "ВВЕДИТЕ ОПИСАНИЕ ФОРМАТА", X⊙(K)
60  NEXT K
70  REM ВВОД ЭЛЕМЕНТОВ МАССИВОВ A( ), B( )
80  MAT INPUT A, B
200  PRINTUS1NG 250
210  FOR K=1 TO 10
220  PRINT C⊙(K);
230  PRINTUSING X⊙(K), A(K); B(K)
240  NEXT K
250  % НАИМЕНОВАНИЕ              1982 Г.   1983 Г.
```

В результате выполнения программы будет напечатано

```
НАИМЕНОВАНИЕ          1982 Г.        1983 Г.
ПОКАЗАТЕЛЬ 1          230.25         240.50
ПОКАЗАТЕЛЬ 2          137            140
ПОКАЗАТЕЛЬ 3          150.6          147.8
 . . . . . .
ПОКАЗАТЕЛЬ 10            1.375           0.678
```

Глава 17. УПРАВЛЕНИЕ ЭКРАНОМ ДИСПЛЕЯ

17.1. Коды управления экраном

Когда пользователь осуществляет ввод информации с клавиатуры, то в какой-то момент курсор будет располагаться на последней строке экрана. Если в этой ситуации нажать клавишу CR/LF, содержимое экрана автоматически переместится на одну строку вверх с потерей первой строки, а внизу экрана появится новая свободная строка. Это создает удобство при вводе программ, а иногда и при вводе или выводе данных. Однако довольно часто возникает необходимость работать с «установившимся» изображением на экране дисплея и иметь возможность с помощью программы контролировать положение курсора. Например, может потребоваться, чтобы каждый запрос на ввод очередной информации печатался в первой строке экрана дисплея, а средняя часть экрана использовалась бы для вывода сообщений в случае ошибочного ввода. Для многих прикладных программ характерна проблема редактирования различных полей (текстовых и числовых) напечатанной на экране дисплея таблицы или ее части. Редактирование заключается в изменении содержимого определенных полей, для чего необходимо управлять положением курсора на экране: перемещать курсор влево, вправо, вверх, вниз, очищать экран или его часть и т. п.

Такое управление осуществляется с помощью специальных шестнадцатеричных кодов (команд управления экраном). Эти команды можно выполнить, используя оператор PRINT, функцией HEX с соответствующими кодами. Ниже приводятся управляющие коды экрана дисплея.

HEX-код	**Выполняемая функция**
01	Установка курсора в левый верхний угол
03	Очистка экрана и установка курсора в левый верхний угол
05	Высвечивание ранее погашенного курсора
06	Гашение курсора с сохранением местоположения курсора
07	Подача звукового сигнала

08	Сдвиг курсора на одну позицию влево
09	Сдвиг курсора на одну позицию вправо
0A	Перемещение курсора на одну строку вниз
0C	Перемещение курсора на одну строку вверх
0D	Перемещение курсора в начало строки
11	Индикация белых символов на темном поле экрана
12	Индикация темных символов на белом поле экрана
85	Перемещение курсора в начало следующей строки

Код HEX (01) обеспечивает установку курсора в верхний левый угол экрана. Содержимое экрана при этом не изменяется, например

```
10  PRINT  HEX(01);
20  GOTO  20
```

Во время выполнения этой программы курсор будет находиться в первой позиции первой строки экрана. Например,

```
10  PRINT  HEX (01);  "ПЕЧАТЬ  ЗАГОЛОВКА  В   1   СТРОКЕ"
```

Результатом работы этой программы будет печать сообщения, заданного в кавычках, в первой строке экрана.

Код HEX(03) обеспечивает установку курсора в верхний левый угол экрана и стирание содержимого экрана. Заметим, что при этом содержимое ОЗУ не изменяется. Оператор PRINT HEX (03) употребляется в тех местах программы, где нужно стереть на экране следы работы предыдущих частей программы.

Код HEX(07) обеспечивает подачу звукового сигнала продолжительностью 0,2 с и используется в тех местах программы, при выполнении которых нужно привлечь внимание пользователя. Если нужно подать сигнал большой длительности, можно увеличить количество кодов, например

```
 10   PPINT   HEX(070707)
```

Код HEX(08) обеспечивает передвижение курсора на одну позицию влево, причем символ слева не стирается. Если курсор стоит в первой позиции строки, то в результате выполнения оператора PRINT HEX(08) курсор устанавливается в последнюю позицию текущей строки.

Код HEX(09) обеспечивает перемещение курсора на одну позицию вправо. В отличие от кода пробела HEX (20), ввод которого с помощью клавиши пробела стирает символ справа от курсора перед его перемещением, код HEX (09) сохраняет этот символ. Поэтому HEX(09) иногда называют «неразрушающим пробелом».

Пример 17.1

```
10  PRINT  "ПЛАН   ВЫПУСКА";   HEX (0808080809)
20  PRINT  "ПЛАН   ВЫПУСКА";   HEX (0808080820)
```

В результате выполнения программы будет напечатано

```
ПЛАН   ВЫПУСКА
ПЛАН   ВЫП   СКА
```

Код HEX(0A) обеспечивает перемещение курсора на одну строку вниз, не изменяя текущую позицию курсора в строке (при этом никакие символы не стираются).

Пример 17.2

```
10  PRINT  "ПРОЕКТ";   HEX(0A);   "ПЛАНА"
```

В результате будет напечатано

```
      ПРОЕКТ
            ПЛАНА
```

Код HEX (0C) обеспечивает перемещение курсора на одну строку вверх, не изменяя текущую позицию курсора в строке (при этом никакие символы не стираются).

Пример 17.3

```
10   PRINT  "ПРОЕКТ";   HEX (0C);   "ПЛАНА"
```

В результате будет напечатано

```
      ПЛАНА
ПРОЕКТ
```

Код HEX(0D), который иногда называется кодом возврата каретки, обеспечивает перемещение курсора в начало текущей строки (в позицию 1), при этом содержимое строки экрана сохраняется.

Код HEX(11) обеспечивает переход к дальнейшему выводу информации в виде светлых элементов на темном поле (негативное изображение). Код HEX (12) обеспечивает переход к выводу информации в виде темных элементов на светлом фоне (позитивное изображение). Эти коды используются для выделения отдельных полей при печати на экране.

Пример 17.4

```
10  DIM A⊙10, B⊙80: A⊙="ЗАГОЛОВОК": B⊙
="ПОКАЗАТЕЛЬ"
20  PRINT HEX(030A0A)
30  PRINT HEX(12); A⊙
40  PRINT B⊙, 5, 10.1, 3, 8
50  PRINT HEX(11)
60  PRINT A⊙
70  PRINT B⊙, 5, 10.1, 3, 8
```

В этом примере позитивным изображением выделены печать заголовка A⊙, а затем строки данных. В строке 50 программы стоит оператор PRINT HEX (11), отменяющий позитивное изображение. Повторная печать заголовка и строки данных в строке 70 осуществляется в негативном изображении.

17,2. Оператор позиционирования курсора

Оператор позиционирования курсора предназначен для установки курсора в любую позицию на экране дисплея. Общая форма оператора:

PRINT AT(<а.в>, <а.в> [,<а.в>])

где первый параметр <а.в.> указывает номер строки, второй — номер позиции в строке, третий — количество стираемых символов на экране дисплея.

Параметр <номер строки> может принимать значения от 1 до 24, <номер позиции> — от 1 до 80. Если указанные в операторе PRINT AT параметры больше предельных значений, то курсор устанавливается по максимально допустимым значениям. Необязательный параметр <количество стираемых символов> задает количество символов экрана дисплея, подлежащих стиранию до установки курсора в заданную позицию. Стираются символы, начиная от определенной первыми двумя параметрами позиции. Если число стираемых символов превосходит количество символов в текущей строке, то стираются символы следующей строки.

Пример 17.5

```
100  PRINT  AT(8, 30)
110  PRINT  "ЗАГОЛОВОК"
```

При выполнении строки 100 курсор устанавливается в тридцатую позицию восьмой строки экрана. Печать слова «ЗАГОЛОВОК» (строка 110) осуществляется, начиная с этой позиции.

Пример 17.6. Ввести с клавиатуры число А. Если А больше 145, то в строке 10 экрана напечатать сообщение о неверном вводе и ввести А заново. Запрос на ввод числа А вывести в строке 8 экрана.

```
100  PRINT AT (8, 20, 15)
110  INPUT A
120  IF A<=145 THEN 200
130  PRINT AT(10, 20)
140  PRINT "НЕВЕРНЫЙ ВВОД - A>145"
150  GOTO 100
200  PRINT AT(10,1,80)
```

В этом примере знак вопроса при выполнении оператора (строка 110) появится в двадцатой позиции восьмой строки экрана. Если введенное значение переменной А больше 145, то в строке 10 экрана будет напечатано сообщение «НЕВЕРНЫЙ ВВОД — A>145», курсор будет установлен в двадцатую позицию восьмой строки экрана, вправо от этой позиции будут стерты пятнадцать символов (результат неверного ввода) и по оператору INPUT введено новое значение А. При вводе допустимого значения переменной А происходит переход к строке 200, в которой с помощью оператора PRINT AT стирается содержимое строки 10 экрана, где, возможно, было сообщение о неверном вводе, и выполнение программы

продолжается.

17.3. Редактирование таблиц

Допустим, требуется написать программу для печати таблицы на экране и корректировки числовых значений в ней. Таблица содержит две графы — «НАИМЕНОВАНИЕ» и «КОЛИЧЕСТВО» и десять строк

```
НАИМЕНОВАНИЕ              ПЛАН
ПОКАЗАТЕЛЬ 1               15.3
 . . . . . . . . . . . . . . . . . . . . . . . . .
ПОКАЗАТЕЛЬ 10              32.0
```

Пусть наименования хранятся в массиве A(), числовые значения— в массиве X().

Пример 17.7

```
10   DIM A⊙(10)20,   X(10)
15   % ####.#
20   REM ВВОД ЗНАЧЕНИЙ МАССИВОВ A⊙( ) и X( )
30   MAT INPUT A⊙( ), X( )
90   REM ОБРАЩЕНИЕ К ПОДПРОГРАММЕ ПЕЧАТИ ТАБЛИЦЫ
100  G0SUB' 50
110  P = 0
120  PRINT AT(15, 30, 40)
130  INPUT "ЕСЛИ ВЕРНО-CR/LF, ПРАВИТЬ - 1", P
140  IF P=0 THEN 170
145  REM ОБРАЩЕНИЕ К ПОДПРОГРАММЕ КОРРЕКТИРОВКИ
150  GOSUB '60
160  GOTO 110
170  END
1000 DEFFN '50: REM ПОДПРОГРАММА ПЕЧАТИ ТАБЛИЦЫ
1010 PRINT HEX(03)
1020 PRINT AT(2, 1)
1030 PRINT "НАИМЕНОВАНИЕ", TAB(25); "  ПЛАН "
1040 FOR  K=1 TO 10
1050 PRINT A⊙ (K); TAB (25);
1060 PRINTUSING 15, X(K)
1070 NEXT K : RETURN
1200 DEFFN '60: REM  ПОДПРОГРАММА КОРРЕКТИРОВКИ
1220 FOR K=1 TO 10
1230 PRINT AT(2+K, 23)
1240 INPUT X(K)
1250 PRINT AT(2+K, 23, 2)
1255 PRINT AT(2+K, 25)
1260 PRINTUSING 15, X(K)
1270 NEXT K : RETURN
```

Программа примера состоит из основной части (строки 10— 500) и двух подпрограмм для печати и корректировки таблицы. Во время исполнения строки 100 происходит обращение к подпрограмме печати таблицы, занимающей строки 1000—1090. Она включает в себя операцию очистки экрана, печать заголовка во 2-й строке экрана и печать 10 строк таблицы. Печать числовых значений осуществляется по формату, заданному в строке 15, начиная с 25-й позиции. Подпрограмма корректировки обеспечивает повторный ввод значений элементов массива X(). Для этого курсор устанавливается в начало числового поля в позицию 25 текущей строки, причем знак вопроса при выполнении оператора INPUT появится в позиции 23. Затем вводится новое значение X(K) (строка 1240), знак вопроса стирается, курсор снова устанавливается в начало числового поля и вновь введенное значение X(K) печатается по формату в строке 15.

Обобщим этот пример, введя в таблицу еще две графы — «ОТЧЕТ» и «ПРОЕКТ». В программу необходимо внести следующие изменения. Во-первых, изменить размерность числового массива X() в строке 10.

```
10  DIM A⊙ (10)  20,  X(10, 3);
```

Во-вторых, изменить подпрограмму печать таблицы.

```
1000 DEFFN  '50:  REM ПОДПРОГРАММА ПЕЧАТИ ТАБЛИЦЫ
1005  % # # # # .#
1010 PRINT  HEX (03)
1020 PRINT  AT (2, 1)
1030 PRINT "НАИМЕНОВАНИЕ", TAB ( 25);"  ПЛАН
ОТЧЕТ  ПРОЕКТ"
1040  FOR  K=1 TO 10
1050   PRINT A⊙ (K); TAB (25);
1055    FOR T=1 TO 3
1060              PRINTUSING 15, X (K,T);
1065    NEXT T
1070 NEXT K
1080 RETURN
```

Для печати числовой строки используется оператор цикла.

В-третьих, изменить подпрограмму корректировки.

```
1200  DEFFN '60: REM  ПОДПРОГРАММА КОРРЕКТИРОВКИ
1220  FOR K=1 TO 10
1225  FOR T=1 TO 3
1230        PRINT AT (2+K, 23+9*(T-1))
1240        INPUT X (K, T)
1250        PRINT AT(2+K, 23+9*(T-1), 2)
1255        PRINT AT (2+K, 25+9*(T-1))
1260        PRINTUSING 15, X (K, T)
1265  NEXT T
1270  NEXT K
1280  RETURN
```

В подпрограмме также введен внутренний цикл по номеру графы Т для ввода новых числовых значений. Номер позиции, в которую устанавливается курсор, вычисляется с помощью выражения как функция номера графы таблицы.

В заключение отметим, что использование оператора INPUT при редактировании таблиц не всегда удобно. Значительно большие возможности в этом случае обеспечивает оператор посимвольного ввода с клавиатуры KEYIN (оператор KEYIN в данной книге не рассматривается).

Глава 18. ОПЕРАЦИИ С ДИСКОВЫМИ ФАЙЛАМИ

18.1. Каталог диска и файлы данных

Как уже отмечалось в гл. 5, операторы Бейсика микроЭВМ «Искра 226» позволяют работать с диском в двух основных режимах: абсолютной адресации секторов и автоматической каталогизации файлов. В режиме абсолютной адресации секторов пользователь непосредственно работает с адресами секторов. В частности, в операторах чтения-записи программ или данных он должен специфицировать номер сектора начала программы или данных. Естественно, что программирование в режиме абсолютной адресации секторов достаточно трудоемко, поскольку пользователь должен сам распределять секторы диска под программы и данные и постоянно помнить это распределение.

Операторы режима автоматической каталогизации файлов обеспечивают большие удобства пользователю при работе с диском. Машина в этом режиме сама осуществляет распределение секторов для хранения данных. В гл. 5 были рассмотрены основные понятия режима автоматической каталогизации файлов и операции с программными файлами на магнитном диске.

При работе с данными в режиме автоматической каталогизации обеспечиваются два иерархических уровня работы: уровень файлов и уровень записей. Под записью понимается элементарная единица данных на диске, обрабатываемая операторами Бейсика. Запись может содержать одно или более значений (т. е. чисел или строк символов) и занимать один или несколько последовательных секторов диска. Отдельные значения записи называются элементом или полем записи.

Под файлом понимается совокупность последовательных записей. В общем случае файл может содержать записи с разным числом элементов и с различным типом этих элементов. Размер файла в секторах и имя файла указываются пользователем при создании файла, при этом в указатель каталога

заносятся имя файла и адреса его начального и конечного секторов.

Таблица 18.1

Оператор	Назначение оператора
DATASAVE DC OPEN	Создает на диске новый файл и подготавливает процессор к операциям с файлами
DATALOAD DC OPEN	Подготавливает процессор к операциям с существующим файлом
DATASAVE DC	Записывает данные из оперативной памяти в файл в виде одной логической записи
DATASAVE DC END	Записывает в файл специальную запись — признак конца данных в файле
DATA LOAD DC	Загружает данные одной записи файла и связывает эти данные с переменными в оперативной памяти
DSKIP DBACKSPACE	Эти операторы изменяют адрес текущего сектора файла. Позволяют осуществить доступ к произвольной записи в файле
LIMITS	Присваивает переменным значениям адресов начального, конечного и текущего секторов файла с указанным именем

В указателе каталога нет информации о расположении отдельных записей файла, и задача нахождения нужной записи внутри файла ложится на программу пользователя.

В табл. 18.1 приводится перечень операторов для работы с файлами данных и их функций.

18.2. Создание и открытие файла данных

Оператор записи программ SAVE обеспечивает выполнение двух различных процедур. Во-первых, он обеспечивает запись в указатель каталога имени программы и адресов начального и конечного секторов той области диска, куда должна быть записана программа. Во-вторых, он обеспечивает запись текста программы в эту область. В случае файлов данных эти две процедуры реализуются с помощью различных операторов.

18.2.1. СОЗДАНИЕ НОВОГО ФАЙЛА ДАННЫХ

Для создания нового файла на диске используется оператор DATA SAVE DC OPEN. Он обеспечивает запись имени создаваемого файла в указатель каталога вместе с адресами начального и конечного секторов файла. При создании нового файла пользователь должен указать имя файла и число секторов, необходимых для размещения данных. При выполнении оператора DATA SAVE DC OPEN машина использует последний сектор файла для хранения служебной информации. Никакая другая информация по этому оператору в файл не записывается.

Оператор

```
DATA SAVE DC OPEN DC F(100) "ДАННЫЕ"
```

создает новый файл с именем «ДАННЫЕ» и отводит на диске 100 секторов, начиная с первого свободного сектора области каталога на правом диске. Имя файла «ДАННЫЕ» вместе с адресами начального и конечного секторов файла заносится в указатель каталога, а в последний сектор созданного файла заносится служебная информация.

Имя файла данных, так же как и в случае программных файлов, может содержать до 8 символов и может задаваться с помощью символьной переменной. Так, оператор

```
100 DATA SAVE DC OPEN DC F(100) "ДАННЫЕ"
```

и оператор

```
100 K⊙ = "ДАННЫЕ"
100 DATA SAVE DC OPEN DC F(100) K⊙
```

полностью эквивалентны. В режиме автоматической каталогизации во всех операторах, требующих

указания имени файла, последний может задаваться с помощью символьной переменной.

Перед тем, как создавать первый файл с помощью оператора DATA SAVE DC OPEN, пользователь должен решить, сколько секторов нужно выделить под создаваемый файл. При этом возникает ряд вопросов:

1. Какая информация будет храниться в каждой записи файла?
2. Сколько секторов требуется для размещения одной записи на диске?
3. Какое максимальное число записей предполагается хранить в файле?

Следует четко помнить, что число секторов, выделяемых для размещения нового файла (в операторе DATA SAVE DC OPEN), должно быть подсчитано точно. Если это число секторов окажется слишком малым, то увеличить область на диске для созданного файла уже нельзя (по крайней мере, элементарными средствами).

Итак, оператор DATASAVE DC OPEN создает новый дисковый файл посредством занесения в указатель каталога имени файла и адресов граничных (начального и конечного) секторов этого файла. Для того чтобы сделать возможным выполнение различных операций с файлом данных в режиме каталога, машина должна иметь адреса трех секторов в специальной области памяти, называемой таблицей устройств. Этими тремя адресами являются адреса начального и конечного секторов файла, а также так называемый адрес текущего сектора. Когда машина записывает в строку таблицы устройств адреса для данного файла, говорят, что файл «открыт», потому что теперь становятся возможными операции записи данных в файл или чтения данных из него.

Оператор DATA SAVE DC OPEN создает новый файл и открывает его, т. е. заносит имя файла и адреса его граничных секторов в указатель каталога, а также записывает адреса начального, конечного и текущего секторов в строку таблицы устройств. Адреса начального и конечного сектора файла в таблице устройств совпадают с адресами, хранящимися в указателе каталога, и указывают системе на фиксирование границы файла на диске. При выполнении оператора DATA SAVE DC OPEN адрес текущего сектора устанавливается равным адресу начального сектора. Этот адрес изменяется в процессе записи или чтения данных из файла.

После включения машины и загрузки внутреннего математического обеспечения в нулевой строке таблицы устройств хранятся нулевые значения.

Начальный адрес	Конечный адрес	Адрес текущего сектора
0	0	0

Пусть параметр «CURRENT END» каталога (см. гл. 5) на правом диске равен 24. Тогда после выполнения оператора

```
:DATA SAVE DC OPEN F(100) "ДАННЫЕ"
```

нулевая строка таблицы устройств будет выглядеть так:

Начальный адрес	Конечный адрес	Адрес текущего сектора
25	124	25

18.2.2. ОТКРЫТИЕ НОВОГО ФАЙЛА НА МЕСТЕ ВЫЧЕРКНУТОГО

Возможны ситуации, когда по истечении некоторого времени данные, хранившиеся в дисковом файле, устаревают и становятся ненужными. В этом случае область диска, занимаемая таким ненужным файлом, может быть использована для размещения в ней нового файла. Для этого нужно сначала вычеркнуть файл из каталога с помощью оператора SCRATCH. Например, оператор

```
SCRATCH R "ДАННЫЕ"
```

отмечает в указателе каталога на левом диске имя файла данных «ДАННЫЕ» как вычеркнутый. После этого на месте, занимаемом файлом «ДАННЫЕ», можно открыть новый файл, например с именем «ДАННЫЕ 2», для чего в операторе создания нового файла нужно указать в скобках вместо числа секторов имя вычеркнутого файла.

```
DATA SAVE DC OPEN F ("ДАННЫЕ") "ДАННЫЕ 2"
```

В результате в указатель каталога имя вычеркнутого файла будет заменено на имя нового файла, а адреса граничных секторов не изменятся. Еще раз отметим, что на месте вычеркнутого файла нельзя открыть новый файл размером больше или меньше первоначального. При открытии нового файла данных на месте вычеркнутого содержимое секторов диска, занимаемых вычеркнутым файлом, не

изменяется.

На месте файла данных можно записать программный файл, и наоборот, например

```
: SCRATCH F "Д1"
```

помечаем как ликвидируемый файл данных с именем «Д1».

```
:SAVE DCF("Д1") "ПРОГ1"
```

На месте файла «Д1» записывается программа с именем «ПРОГ 1».

18.2.3. ОТКРЫТИЕ ФАЙЛА, СОЗДАННОГО РАНЕЕ

Оператор DATA SAVE DC OPEN используется только для создания нового файла, после чего имя этого файла и адреса

его граничных секторов постоянно хранятся в указателе каталога. Для работы с уже созданным файлом необходимо лишь считать адреса его граничных секторов в таблицу устройств. Такая операция («открытие» уже существующего файла) осуществляется с помощью оператора DATA LOAD DC OPEN. Например, по оператору

```
DATA LOAD DC OPEN F "ДАННЫЕ"
```

выполняется поиск в указателе каталога имени файла «ДАННЫЕ». Если имя находится, в таблицу устройств считываются адреса граничных секторов файла с именем «ДАННЫЕ», а адрес текущего сектора устанавливается равным адресу начального сектора. Начинающие пользователи часто смешивают операторы DATASAVE DC OPEN и DATA LOAD DC OPEN, поэтому приводим подробное сравнение их функций (табл. 18.2).

Таблица 18.2

Оператор	Назначение оператора
DATASAVE DC OPEN	Создает новый файл данных заданного размера путем включения имени файла и адресов граничных секторов в указатель каталога; открывает новый файл, чтобы обеспечить возможности записи (чтения данных из файла с помощью дисковых операторов в режиме каталога). Адреса граничных секторов файла заносятся в таблицу устройств, адрес текущего сектора устанавливается равным адресу начального
DATALOAD DC OPEN	Только открывает файл посредством занесения адресов граничных секторов файла из указателя каталога в таблицу устройств. Адрес текущего сектора устанавливается равным адресу начального сектора

После открытия файла данные можно записывать в него или читать из него независимо от того, с помощью какого оператора (DATA SAVE DC OPEN или DATA LOAD DC OPEN) был открыт этот файл.

18.3. Запись данных в файл

Следующим шагом после создания и открытия нового файла с помощью оператора DATA SAVE DC OPEN является процедура записи данных в этот файл. Для записи данных в каталогизированный файл используется оператор DATA SAVE DC. Например, оператор

```
DATA SAVE DC A⊙, B⊙, C⊙, D, E
```

обеспечивает запись текущих значений переменных A⊙, B⊙, C⊙, D и E в файл, начиная с сектора, адрес которого берется из графы таблицы устройств «адрес текущего сектора». По окончании записи данных адрес текущего сектора в таблице устройств изменяется на адрес сектора, следующего за последним сектором, занятым под данные. Совокупность значений, записываемых с помощью одного оператора DATA SAVE DC, называется логической записью данных, или просто записью, а каждое значение—элементом или полем записи. Записываемые с помощью оператора DATA SAVE DC данные

могут занять один или несколько секторов. Помимо вывода самих значений указанных в операторе DATA SAVE DC переменных, машина записывает определенную служебную информацию, которая позволяет при считывании различать записанные значения. Таким образом, результатом выполнения оператора DATA SAVE DC является создание одной логической записи в файле данных. Перед тем как выполняется оператор DATA SAVE DC, в программе должны быть определены все величины, указанные в этом операторе. Предположим, что оператор

```
DATA SAVE DC A⊙, B⊙, C⊙, D, E
```

создает запись в файле анкет. Пусть в A⊙ хранится учетный номер работника, в B⊙ — его фамилия, в C⊙ — наименование профессии, в D — год поступления на работу, в E — размер должностного оклада. Эти величины составляют вместе полную запись о работнике в анкетном файле предприятия. Следующая программа демонстрирует пример создания нового анкетного файла и записи в него ряда записей — анкет.

Пример 18.1

```
100  REM ОПИСАНИЕ ЭЛЕМЕНТОВ ЗАПИСИ
110  DIM A⊙5, B⊙20, C⊙20
120  REM СОЗДАНИЕ И ОТКРЫТИЕ НОВОГО ФАЙЛА
130  DATA SAVE DC OPEN F (250) "АНКЕТА"
140  REM ВВОД ДАННЫХ ДЛЯ ОДНОЙ ЗАПИСИ (АНКЕТЫ)
150  INPUT "УЧЕТНЫЙ НОМЕР", A⊙
160  INPUT "ФАМИЛИЯ", B⊙
170  INPUT "ПРОФЕССИЯ", C⊙
180  INPUT "ГОД ПОСТУПЛЕНИЯ",D
190  INPUT "ОКЛАД", E
200  REM ЗАПИСЬ НА ДИСК
210  DATA SAVE DC A⊙, B⊙, C⊙ D, B
220  REM БУДУТ ЕЩЕ ЗАПИСИ?
230  INPUT "БУДУТ ЕЩЕ ЗАПИСИ (ДА - 1, НЕТ - 2)", P
240  IF P = 1 THEN 150
250  END
```

В строке 130 создается и открывается новый файл с именем «АНКЕТА» размером в 250 секторов. В строках 150—190 вводятся значения переменных — элементов записи. В строке 210 данные записываются в файл, начиная с сектора, адрес которого равен адресу текущего сектора из таблицы устройств. Каждый раз после очередной записи данных адрес текущего сектора в таблице устройств устанавливается равным адресу сектора, следующего за последним занятым данным сектором. После открытия файла адрес текущего сектора равен адресу начального сектора файла. Первая запись (анкета) будет размещена в этом секторе. Каждая последующая запись будет размещена в следующем секторе. Таким образом, порядок размещения записей — анкет на диске будет соответствовать порядку их ввода.

Повторное выполнение программ из примера 18.1 невозможно. Попытка создать новый файл с именем «АНКЕТА» (строка 130) приведет к останову по ошибке, так как в указателе каталога уже есть такое имя. Для повторного открытия существующего файла необходимо воспользоваться оператором DATA LOAD DC OPEN.

В примере элементами списка оператора DATASAVE DC являются символьные и числовые переменные. Элементами оператора записи могут быть также числовые или символьные массивы. В этом случае их имена указываются в операторе DATA SAVE DC вместе с круглыми скобками, например A(), B⊙ () и т. д. Элементами записи могут быть также символьные константы и выражения. В случае выражений на диск записываются их значения.

Пример 18.2

```
100 DATA SAVE DC "ПРОЕКТ ПЛАНА" P⊙ ( ), K
120 DATA SAVE DC D+X*Z/A, E ( )
```

Внутри записи на диске элементы записи размещаются последовательно в том порядке, в каком они указаны в операторе DATA SAVE DC. Массивы записываются строка за строкой. Перед каждым значением (числовым или символьным) записываются два служебных символа, отделяющие одно значение от другого и содержащие информацию о типе (числовое или символьное) и длине значения. Следует помнить, что по оператору DATA SAVE DC на диск выводятся только значения переменных,

но не их имена.

18.4. Запись признака конца данных в файле и закрытие файла

Когда создается новый файл, его размеры рассчитываются пользователем исходя из оценки максимально возможного числа записей. В большинстве случаев файлы могут содержать некоторое число неиспользованных секторов между последней на данный момент записью данных в файле и концом (последним сектором) файла. Во многих случаях удобно с помощью записи специального вида помечать текущий конец данных в файле для того, чтобы такая концевая запись определяла сектор, откуда можно записывать данные.

Оператор DATA SAVE DC END используется для занесения в файл специальной односекторной записи, идентифицирующей конец данных в файле. Эта запись помещается в сектор с адресом, равным адресу текущего сектора в таблице устройств.

Для того чтобы в программе примера 18.1 включить в файл «АНКЕТА» концевую запись, нужно добавить строку

```
245 DATA SAVE DC END
```

Идентификация текущего конца данных в файле не является необходимостью, однако желательна. Использование оператора DATA SAVE DC END помогает пользователю визуально контролировать число использованных секторов в файле при выполнении оператора LIST DC. Если признак конца данных в файле не записан, то в графе «Использовано» для такого файла всегда будет стоять 00001. Если же признак конца данных в файле записан, то в этой графе будет стоять число секторов, реально использованных для записи данных.

Процедуру открытия файла можно определить просто как запись адресной информации о файле в таблицу устройств. Открытый файл может быть обработан с помощью дисковых операторов в режиме каталога. После завершения обработки файл может быть закрыт посредством открытия другого файла. Файл может быть закрыт также при выполнении команды CLEAR либо при повторной инициализации системы. Так как в закрытый файл невозможно записывать данные, то закрытие файла в общем случае предохраняет его от случайного разрушения при выполнении других подпрограмм или дисковых операций в режиме непосредственного счета. Оператор DATA SAVE DC CLOSE закрывает файл, записывая нули во все графы таблицы устройств. Закрытый с помощью оператора DATA SAVE DC CLOSE файл может быть повторно открыт с помощью оператора DATA LOAD DC OPEN.

Возвращаясь к примеру 18.1, добавим в конце программы оператор закрытия файла «АНКЕТА».

```
247 DATA SAVE DC CLOSE
```

18.5. Загрузка данных из файла

После того как файл создан и в нем размещены несколько записей данных, последние могут быть загружены (считаны) с Диска в оперативную память. Для того чтобы загрузить запись в режиме каталога, файл должен быть открыт, т. е. адреса начального, конечного и текущего секторов файла должны быть Помещены в таблицу устройств. Созданный ранее файл всегда открывается с помощью оператора DATA LOAD DC OPEN. Для загрузки данных из файла используется оператор DATA LOAD DC. Например, оператор

```
DATA LOAD DC A⊙, B⊙, C⊙, D, E
```

обеспечивает загрузку данных, начиная с сектора, адрес которого указан как адрес текущего сектора в таблице устройств. Считанные значения присваиваются последовательно переменным, которые указаны в операторе DATA LOAD DC. По окончании операции загрузки данных адрес текущего сектора в таблице устройств будет равен адресу первого сектора следующей записи файла. Присвоение переменным значений во время операции загрузки происходит с учетом следующих правил. Если символьная переменная короче символьного значения, то лишние символы справа отбрасываются. При попытке прочитать символьное значение в числовую переменную и, наоборот, числовое значение в символьную переменную происходит останов по ошибке. Возможность возникновения таких ситуаций требует тщательного документирования структуры записей файлов.

Записанные значения могут быть загружены в массив, для чего имя массива должно быть указано в операторе DATA LOAD DC. Например, оператор

```
DATA  LOAD  DC P( ), T( )
```

загружает данные с диска в массив P() строка за строкой до тех пор, пока все элементы массива P() не будут полностью заполнены, а остальные данные оператор загружает в массив T().

При загрузке данных по оператору DATA LOAD DC в нем могут быть указаны имена переменных, отличные от тех, которые использовались при записи данных по оператору DATA, SAVE DC. Единственное требование здесь — числовые значения должны загружаться в числовые переменные, символьные значения — в символьные переменные. Следующая программа демонстрирует загрузку и печать данных из файла, созданного программой примера 18.1.

Пример 18.3

```
100  DIM A⊙5, B⊙20, C⊙20
110  SELECT PRINT /OC
120  REM ОТКРЫТИЕ ФАЙЛА
130  DATA LOAD DC OPEN F "АНКЕТА"
140  DATA LOAD DC A⊙, B⊙, C⊙, D, E
150  PRINT A⊙, B⊙, C⊙, D, E
160  GOTO 140
```

В строке 100 размерности символьных переменных такие же, как и в примере 18.1, где выполнялась запись данных в файл «АНКЕТА».

Строки 140—160 формируют цикл, на каждом шаге которого читается и распечатывается на АЦПУ очередная запись. На каком-то шаге цикла либо будут исчерпаны все записи файла, либо адрес текущего сектора сравняется с адресом конечного сектора файла. В этом случае произойдет останов по ошибке, означающий, что оператор DATA LOAD DC пытается загрузить несуществующую запись.

В Бейсике «Искры 226» есть специальный оператор для обнаружения факта чтения концевой (или закрывающей) записи файла оператором DATA LOAD DC. Он имеет следующий вид:

```
IF  END  THEN <номер  строки>
```

При считывании концевой записи по оператору DATA LOAD DC происходит переход к строке с номером, указанным в операторе IF END THEN. Изменим программу из примера 18.3, включив оператор IF END THEN в цикл.

Пример 18.4

```
100  DIM A⊙5, B⊙20, C⊙20
110  SELECT PRINT /OC
120  REM ОТКРЫТИЕ ФАЙЛА
130  DATA LOAD DC OPEN F "АНКЕТА"
140  DATA LOAD DC A⊙, B⊙, C⊙, D, E
145  IF END THEN 170
150  PRINT A⊙, B⊙, C⊙, D, E
160  GOTO 140
170  DATA SAVE DC CLOSE
```

Оператор IF END THEN в строке 145 обеспечивает переход к строке 170 в том случае, когда оператор DATA LOAD DC считает концевую запись. Поэтому останова по ошибке уже не будет, программа может выполняться дальше, если это необходимо. Еще раз отметим, что описанным способом можно проверить наличие специальной концевой записи, созданной с помощью оператора DATA SAVE DC END.

18.6. Доступ к отдельным записям файла

В разделах 18.3 и 18.4 рассматривались простейшие процедуры обработки файлов, представляющие собой либо последовательную запись данных в файл, либо последовательную загрузку записей файла. Многие практически важные задачи требуют именно такой последовательной обработки, и она легко реализуется с помощью операторов чтения-записи DATA LOAD DC, DATA SAVE DC. Во многих задачах часто возникает необходимость загрузки или обновления заранее заданной записи. Проблема доступа к нужной записи заключается в определении адреса сектора, начиная с которого нужная запись размещается на диске. В качестве примера рассмотрим две задачи:

найти положение концевой записи в файле «АНКЕТА» для того, чтобы иметь возможность добавить в файл несколько записей;

найти запись с номером N в этом же файле для того, чтобы изменить ее содержимое (например,

исправить наименование профессии или размер оклада).

П р и м е р 18.5. Определение положения концевой записи.

```
100  DIM A⊙5, B⊙20, C⊙20
110  REM ОТКРЫТИЕ ФАЙЛА
120  DATA LOAD DC OPEN F "АНКЕТА"
130  DATA LOAD DC A⊙, B⊙, C⊙, D, E
140  IF END THEN 180
150  GOTO 130
160  REM КОНЦЕВАЯ ЗАПИСЬ НАЙДЕНА, т.е. АДРЕС
170  REM "ТЕКУЩЕГО АДРЕСА РАВЕН АДРЕСУ КОНЦЕВОЙ
ЗАПИСИ
180  DATA SAVE DC CLOSE
```

В этом примере запрограммирована последовательная загрузка записей файла «АНКЕТА» до тех пор, пока не будет обнаружена концевая запись.

П р и м е р 18.6. Определение положения записи с номером N.

```
100  DIM  A⊙5,  B⊙20,  C⊙20
110  REM ОТКРЫТИЕ ФАЙЛА
120  DATA LOAD DC  OPEN  F "АНКЕТА"
130  FOR K=1  TO N-1
140  DATA LOAD DC A⊙,  B⊙,  C⊙,  D,  E
150  NEXT K
160  REM АДРЕС  ТЕКУЩЕГО  СЕКТОРА РАВЕН АДРЕСУ
170  REM НАЧАЛЬНОГО  СЕКТОРА  ЗАПИСИ  С  НОМЕРОМ
```

В обоих примерах для доступа к нужной записи (в первом случае — концевой, во втором — с номером N) используется метод последовательной загрузки определенного числа записей. Каждая такая загрузка изменяет адрес текущего сектора таблицы устройств до тех пор, пока не будет получен адрес начального сектора нужной записи. В Бейсике «Искры 226» есть два оператора, которые непосредственно могут изменять адрес текущего сектора: DSKIP и DBACKSPASE. Каждый из этих операторов имеет три формы, описанные в табл. 18.3.

Использование оператора DSKIP END имеет смысл, когда файл данных содержит концевую запись. Оператор DSKIP END

Т а б л и ц а 18.3

Оператор	Назначение оператора
DSKIP <a. в.> S	Целая часть указанного выражения прибавляется к адресу текущего сектора
DSKIP END	Адрес текущего сектора устанавливается равным адресу сектора концевой записи
DSKIP <a. в.>	Пропускается число записей, равное целой части выражении
DBACKSPACE <a. в.> S	Целая часть указанного выражения вычитается из адреса текущего сектора
DBACKSPACE BEG	Адрес текущего сектора устанавливается равным адресу начального сектора файла
DBACKSPACE <a. в.>	Пропускается назад число записей, равное целой части выражения

выполняется значительно быстрее, чем процедура, запрограммированная в примере 18.5.

Операторы DSKIP и DBACKSPACE просто добавляют или вычитают соответственно целую часть указанного в этих операторах выражения к адресу или из адреса текущего сектора. Величина выражения должна быть положительной. Если новое значение адреса текущего сектора в результате выполнения оператора DBACKSPACE становится меньше адреса начального сектора файла, то адрес текущего сектора устанавливается равным адресу начального сектора файла. Аналогично, если адрес текущего сектора в результате выполнения оператора DSKIP становится больше адреса конечного адреса файла, то адрес конечного сектора записывается на место адреса текущего сектора. Таким образом, пользуясь операторами DATA LOAD DC, DATA SAVE DC для последовательного доступа к записям файла и операторами DSKIP и DBACKSPACE для прямого доступа, невозможно выйти за пределы файла и случайно разрушить данные соседних файлов.

Использование операторов DSKIP S, DBACKSPACE S — наиболее эффективный способ изменения

адреса текущего сектора, так как он не требует выполнения каких-либо операций самим дисковым устройством. При этом необходимо учитывать два момента. Во-первых, используя оператор DSKIP S, можно случайно проскочить текущую концевую запись файла и нарушить последовательный порядок размещения записей в файле. Во-вторых, использование этих операторов предполагает знание длины записи файла, выраженной в секторах. Пусть, например, длина записи файла — 3 сектора. Чтобы загрузить 16-ю запись, необходимо пропустить 15 записей или 15 секторов с помощью оператора DSKIP 45 S. Следует отметить, что в большинстве задач длина записи известна заранее при проектировании структуры файла.

Если размер записи неизвестен или файл состоит из записей различной длины, для пропуска «вперед» или «назад» нужного тела записей удобно пользоваться операторами DSKIP к DBACKSPACE без параметра S. При выполнении этих операторов дисковое устройство последовательно считывает промежуточные секторы для того, чтобы по специальным меткам конца записи подсчитать нужное число записей.

В следующем примере приведена программа для добавления записей в файл «АНКЕТА». Наложим дополнительное ограничение на элемент записи учетный номер и потребуем, чтобы у каждой вновь создаваемой записи этот элемент был больше, чем у предыдущей записи.

Пример 18.7

```
100  DIM A⊙5, A2⊙5, B⊙20, C⊙20
110  REM ОТКРЫТИЕ ФАЙЛА
120  DATA LOAD DC OPEN F "АНКЕТА"
130  REM ЗАГРУЗКА ПОСЛЕДНЕЙ ЗАПИСИ
140  DSKIP END
150  DBACKSPACE 1S
160  DATA LOAD DC A2⊙, B⊙, C⊙, D, E
170  REM A2⊙ - УЧЕТНЫЙ НОМЕР ИЗ ПОСЛЕДНЕЙ АНКЕТЫ
180  REM      ВВОД НОВОЙ ЗАПИСИ
190  PRINT HEX(03): "ЗАКОНЧИТЬ ВВОД - КЛЮЧ 15"
200  INPUT "ВВЕДИТЕ НОВЫЙ УЧЕТНЫЙ НОМЕР", A⊙
210  IF NUM(A⊙)<>5 THEN 340
220  IF A⊙<=A2 THEN 340
230  REM УЧЕТНЫЙ НОМЕР ВВЕДЕН ПРАВИЛЬНО
240  INPUT "ФАМИЛИЯ", B⊙
250  INPUT "ПРОФЕССИЯ" C⊙
260  INPUT "ГОД ПОСТУПЛЕНИЯ", D
270  INPUT "ОКЛАД", E
280  REM ЗАПИСЬ НОВОЙ АНКЕТЫ
290  DATA SAVE DC A2⊙, B⊙, C⊙, D, E
300  A2⊙ = A⊙
310  PRINT
320  GOTO 200
330  REM НЕВЕРНЫЙ УЧЕТНЫЙ НОМЕР
340  PRINT "НЕВЕРНЫЙ УЧЕТНЫЙ НОМЕР"
350  PRINT
360  GOTO 200
370  KEM ВЫХОД ИЗ ПРОГРАММЫ
380  DEFFN ' 15
390  DATA SAVE DC END
400  DATA SAVE DC CLOSE
```

В строке 140 адрес текущего сектора устанавливается равным адресу сектора концевой записи. Для того чтобы прочитать последнюю запись файла, в строке 150 записан оператор DBACK.SPACE, обеспечивающий «пропуск назад» одного сектора. Эта запись затем загружается для того, чтобы получить ее элемент— пятисимвольный учетный номер (A2⊙). Знание величины этого элемента необходимо для контроля при вводе элементов следующей записи. Вновь введенный учетный номер A⊙ сравнивается с учетным номером предыдущей записи A2⊙. Если A⊙ меньше A2⊙, то происходит переход к строке 340, выдается сообщение об ошибке (строки 340—360) и снова запрашивается учетный номер. Попутно в строке 210 проверяется содержимое A⊙ — все байты этой переменной должны быть цифрами. Если введенное значение учетного номера удовлетворяет ограничениям задачи, то запрашиваются значения остальных элементов (полей) записи, запись выводится на диск (строка 290), переменной A2⊙ присваивается значение последнего записанного учетного номера и осуществляется

переход к строке 200 на ввод новой записи (анкеты). Для прекращения ввода нужно в любой момент нажать ключ специальных функций с номером 15. В этом случае выполнится подпрограмма (строки 380—400), обеспечивающая запись концевой записи и закрытие файла.

Наложим еще одно ограничение на учетные номера. Предположим, что они изменяются непрерывно от строки символов «00001» до строки символов, изображающей число записей в файле. Составим программу для изменения одного из полей, например «ОКЛАД», в заранее заданной записи. Задание записи осуществляется с помощью поля «учетный номер».

Пример 18.8

```
100  DIM A⊙5, B⊙20, C⊙20
110  REM ОТКРЫТИЕ ФАЙЛА
120  DATA L9AD DC 0PENF "АНКЕТА"
130  REM ЗАГРУЗКА ПОСЛЕДНЕЙ ЗАПИСИ
140  DSKIP END
150  DBACKSPACE IS
160  DATA LQAD DC A⊙, B⊙, C⊙, D, E
170  CONVERT A⊙ TO  M
180  REM M - МАКСИМАЛЬНЫЙ УЧЕТНЫЙ НОМЕР В ФАЙЛЕ
190  REM ЗАДАНИЕ ЗАПИСИ
200  PRINT HEX(03)
210  INPUT "ВВЕДИТЕ УЧЕТНЫЙ НОМЕР ЗАПИСИ,
ПОДЛЕЖАЩЕЙ ПРАВКЕ", A⊙
220  IF NUM(A⊙) < > 5 THEN 400
230  CONVERT A⊙ TO N
240  IF N > M  THEN 400
250  DBACKSPACE BEG
260  DSKIP N-l
270  REM ЗАГРУЗКА ЗАДАННОЙ ЗАПИСИ
280  DATA LOAD DC A⊙, B⊙, C⊙, D.E
290  PRINT "ОКЛАД="; E
300  DBACKSPACE 1
310  REM ВВОД НОВОГО ЗНАЧЕНИЯ "ОКЛАД"
320  INPUT "ОКЛАД", E
330  DATA SAVE DC A⊙, B⊙, C⊙, D, E
340  P⊙=" "
350  INPUT "ЕСТЬ ЕЩЕ ЗАПИСИ ДЛЯ ПРАВОК (ДА/НЕТ)",
P⊙
360  IF P⊙ = "ДА" THEN 200
370  IF  P⊙ <  > "НЕТ"  THEN  350:REM  ОШИБКА  ВВОДА
380  DATA SAVE DC CLOSE
390  END
400  REM ОБРАБОТКА ОШИБОК
410  PRINT "НЕВЕРНЫЙ УЧЕТНЫЙ НОМЕР"
420  PRINT
430  GOTO 210
```

В строках 130—170 загружается последняя запись файла, и учетный номер из этой записи преобразуется в числовую переменную M для контроля учетного номера записи, подлежащей корректировке (строка 240). Если введенный учетный номер (строка 210) больше M или не является правильным изображением числа, то делается переход на строки 400—430, выдается сообщение об ошибке, и учетный номер запрашивается снова. Если учетный номер введен правильно, то осуществляется доступ к нужной записи (строки 250—260), она загружается, и в ней изменяется поле «оклад». Чтобы вывести измененную запись на диск, пропускается назад один сектор. В строках 340—380 программа ветвится по значению переменной P⊙ либо на ввод учетного номера еще одной записи, либо на окончание программы. В этом примере для того, чтобы изменить одно поле в записи, приходится загружать с диска запись целиком и так же целиком выводить ее на диск по оператору DATA SAVE DC после того, как нужное поле (или элемент записи) было изменено.

18.7. Размещение данных на диске

При создании нового файла пользователь должен указать в операторе DATASAVE DC OPEN

максимальный размер создаваемого файла в секторах. Поэтому, прежде чем приступить к составлению программы обработки файла, нужно сначала пошить два вопроса — какое максимальное число записей должно быть в файле и сколько секторов на диске займет каждая запись. Как уже отмечалось, увеличить размеры созданного файла невозможно. В таких случаях приходится создавать другой файл большего размера, чем созданный, и переписывать содержимое первого файла во второй. При расчете размера файла следует помнить, что при создании файла последний сектор файла используется для хранения служебной информации. Рекомендуется также при работе с файлом всегда использовать концевую запись (или запись признака конца файла), создаваемую с помощью оператора DATA SAVE DC END. Концевая запись занимает один сектор. Таким образом, общий размер файла в секторах должен быть на 2 сектора больше того количества секторов, которые необходимы непосредственно для хранения данных. Рассмотрим теперь вопрос, как размещаются на диске элементы записи, выводимые на диск по оператору DATA SAVE DC. Запись может занимать на диске один или несколько секторов размером по 256 байт каждый. При работе в режиме каталога, кроме значений элементов полей записи, в каждом из этих секторов автоматически записывается служебная информация двух видов:
идентификатор сектора—1 байт в начале сектора;

идентификаторы значений — по 2 байта на каждое значение — содержат информацию о типе значения (цифровое или символьное) и его длине. Таким образом, в каждом секторе для размещения данных могут быть использованы 255 байт.

В качестве иллюстрации рассмотрим запись одной анкеты из предыдущих примеров. Запись ее на диск осуществляется с помощью оператора

```
DATA SAVE DC A⊙, B⊙, C⊙, D, E
```

Элементы записи были описаны в операторе DIM.

```
DIM A⊙5, B⊙20, C⊙20
```

Указанная запись займет в секторе 71 байт. Это число получается суммированием длин значений элементов записи и их служебной информации: 71 = (2+5) +(2+20) +(2+20) +(2+8) + + (2+8). 1 байт используется для записи идентификатора сектора. Остальные 184 байта (184=256—1—71) не используются. Следующая запись данных выводится на диск в следующий сектор и т. д.
Перечислим правила для расчета размера записи.

В каждом секторе для хранения данных могут использоваться 255 байт из общего количества в 256 байт: 1 байт используется под служебную информацию.

Каждое целое число требует для своего размещения 4 байта: 2 байта занимает само значение и 2 байта — служебная информация для идентификации значения. В один сектор могут быть записаны 63 целых числа.

Каждое действительное число требует для своего размещения 10 байт: 8 байт занимает само значение и 2 байта — служебная информация для идентификации значения. В один сектор могут быть записаны 25 действительных чисел.

Каждое символьное значение требует для своего хранения столько байтов, какова длина этого значения, плюс 2 служебных байта. Длина символьного значения описывается в операторах к COM, задается по умолчанию либо числом символов в кавычках в случае символьной константы.

Если значение не помещается в секторе полностью, то оно целиком записывается в следующий сектор. В случае записи на диск массивов элементы массива записываются построчно.

Пример 18.9

```
100 DIM A (6, 5)
200 DATA SAVE DC A( )
```

Массив A() содержит 30 элементов, которые будут выводиться в следующем порядке:

```
A(1,1), A(1,2).....A(1,6), A(2,1),..., A(6,5)
```

Заметим, что при выводе массив A() займет два сектора: в первом запишутся элементы с A (1,1) по A (5,5) включительно, во втором-—оставшиеся 5 элементов с A(6,1) по A(6,5).

При проектировании структуры записи нужно следить за тем, чтобы пространство на диске использовалось как можно эффективнее, т. е. чтобы в каждом секторе число неиспользуемых байтов было как можно меньше. При записи, занимающей несколько секторов, такая экономия дисковой памяти может быть достигнута простым изменением порядка элементов записи, указанных в операторе

DATA SAVE DC.

Пример 18.10

```
100 DIM A⊙(5)50, B⊙(3)64, C⊙43
200 DATA SAVE DC A⊙ ( ), B⊙( ), C
```

Таблица 18.4

Сектор № 1	A⊙ (1)	A⊙ (2)	A⊙ (3)	A⊙ (4)	Не используется
	52 байта	52 байта	52 байта	52 байта	47 байт
Сектор № 2	A⊙ (5)	B⊙ (1)	B⊙ (1)	B⊙ (3)	Не используется
	52 байта	66 байт	66 байт	66 байт	5 байт
Сектор № 3	C⊙	Не используется			
	55 байт	210 байт			

Запись, создаваемая оператором DATA SAVE DC в строке 200, займет на диске 3 сектора, как это показано в табл. 18.4.

Всего в этих трех секторах не используются 262 байта (262 = = 47+5+210). Однако, изменив порядок элементов в операторе DATA SAVE DC, можно разместить ту же запись всего лишь в двух секторах.

```
200 DATA SAVE DC C⊙, A⊙( ), B⊙( )
```

Данные будут размещаться так, как показано в табл. 18.5.

Таблица 18.5

Сектор № 1	A⊙ (1)	A⊙ (2)	A⊙ (3)	A⊙ (4)	Не используется
	45байта	52 байта	52 байта	52 байта	2 байт
Сектор № 2	A⊙ (5)	B⊙ (1)	B⊙ (1)	B⊙ (3)	Не используется
	52 байта	66 байт	66 байт	66 байт	5 байт

Рассмотрим теперь случай, когда запись размещается в одном секторе, используя при этом лишь небольшую часть сектора, как это было в случае файла «АНКЕТА» из предыдущих примеров.

Каждая запись этого файла требует, как было подсчитано, 71 байт для своего размещения в секторе. 184 байта каждого сектора остаются при этом неиспользованными. Такое нерациональное использование дисковой памяти ведет к увеличению размеров файла и времени на его обработку. Для того чтобы эффективно использовать диск, используется прием, называемый блокированием записей. Известно, что один оператор создает на диске одну запись, содержащую в нашем случае информацию о работнике предприятия (анкету), и эта запись занимает один сектор. Невозможно вывести в один сектор несколько записей, но можно в одном операторе DATA SAVE DC объединить несколько анкет. Такой оператор создает на диске тоже одну запись, при загрузке которой программным путем ее легко можно разделить на отдельные анкеты. В примере с файлом «АНКЕТА» в один сектор можно разместить три анкеты. Тогда подобная запись займет 3*71 =223 байта в каждом секторе.

Объединение трех анкет в одном операторе DATA SAVE DC может быть выполнено с помощью перечисления в нем трех групп переменных.

Пример 18.11

```
100 DIM A1⊙5, B1⊙20, C1⊙20, A2⊙5, B2⊙20, C2⊙20, A3⊙5, B3⊙20, C3⊙20
200 DATA SAVE DC A1⊙, B1⊙, C1⊙, D1, E1, A2⊙, B2⊙, C2⊙, D2, E2, A3⊙, B3⊙, C3⊙, D3, E3
```

В результате выполнения строки 200 будет создана одна запись на диске, содержащая три анкеты. Эти анкеты связаны с соответствующими группами переменных так:

	Учетный номер	Фамилия	Должность	Год поступления	Оклад
Анкета 1	A1⊙	B1⊙	C1⊙	D1	E1
Анкета 2	A2⊙	B2⊙	C2⊙	D2	E2
Анкета 3	A3⊙	B3⊙	C3⊙	D3	E3

Возможен другой способ объединения трех анкет в одну запись на диске.

Пример 18.12.

```
100 DIM A⊙(3)5, B⊙(3)20, C⊙(3)20, D(3), E(3)
```

```
200  DATA  SAVE  DC  A⊙( ),  B⊙( ),  C⊙ ( ),  D( ),  E( )
```

В этом случае одинаковые поля групп анкет объединяются (блокируются) в соответствующие массивы. Анкеты могут быть загружены в память машины с помощью оператора

```
DATA  LOAD  DC  A⊙( ),  B⊙( ),  C⊙( ),  D( ),  E( )
```

18.8. Одновременная работа с несколькими файлами

18.8.1. ТАБЛИЦА УСТРОЙСТВ

В предыдущих разделах этой главы рассматривались различные примеры обработки дискового файла. Для того чтобы сделать возможной обработку записей в файле, необходимо сначала открыть этот файл. Операция открытия файла обеспечивает запись адресов граничных секторов файла в таблицу устройств и установку адреса текущего сектора. Открытие другого файла приводит к тому, что в таблицу устройств записывается адресная информация этого файла, а адреса первого файла стираются. Между тем большинство задач требует одновременной обработки нескольких файлов. В качестве примера можно привести перезапись данных из одного файла в другой.

В «Искре 226» возможность одновременной работы с несколькими файлами обеспечивается с помощью таблицы устройств. Ряд сведений о таблице устройств приводился в гл. 6 и 11. Таблица устройств представляет собой специальную область памяти для управления операциями ввода-вывода. Таблицу устройств можно напечатать с помощью оператора LIST %, например

```
   C1 = 01   CO=05(80)   PRINT = 05(80)   LIST=05(80)   TAPE = 08(256)
PLOT=10
   #0—18F
   #1–
   . . .
   #7–
```

Каждая из нижних восьми строк может быть определенным образом связана с дисковым устройством и файлом, размещенным на этом устройстве. Строки нумеруются от 0 до 7 и могут содержать следующую информацию:
физический адрес дискового устройства и тип диска (F или R);
адрес начального сектора файла;
адрес конечного сектора файла;
адрес текущего сектора файла.

При работе с файлом данных последний может быть открыт таким образом, что его адресная информация, ФАУ и тип диска, на котором размещен этот файл, будут храниться в определенной строке таблицы устройств. Все последующие операции обработки этого файла могут выполняться независимо от операций обработки другого файла, связанного с другой строкой таблицы устройств. При работе с несколькими файлами в операторах работы с диском необходима ссылка на номера строк таблицы устройств, в которых хранится информация об этих файлах. Связь конкретного файла со строкой таблицы устройств происходит следующим образом. Сначала с помощью оператора SELECT в строку таблицы устройств заносится ФАУ устройства и тип диска.

Пример 18.17

```
100  SELECT  #2   18F
110  SELECT  #4   1CR,  #5   18R,  #6   18F
120  SELECT   #0   1CR
```

В приведенном операторе SELECT символами # К обозначается номер *к*-й строки таблицы устройств. Этот параметр называется логическим номером устройства или файла, если этот файл связывается с *к*-й строкой таблицы устройств. Такая связь конкретного файла со строкой таблицы устройств осуществляется с помощью операторов DATASAVE DC OPEN и DATA LOAD DC OPEN. Так, оператор

```
 DATA SAVE DC OPEN F#2, "АНКЕТА"
```

открывает файл с именем «АНКЕТА» на правом диске с ФАУ, равным 18, так как этому устройству был назначен логический номер #2. В результате открытия файла «АНКЕТА» адресная информация этого файла будет помещена во вторую строку таблицы устройств. Тип диска устанавливается в самом

операторе открытия файла. Если в операторе открытия файла указан тип диска, отличный от того типа, который был назначен по оператору SELECT, то тип диска берется из оператора открытия файла.

Обработка записей открытого таким способом файла должна осуществляться операторами записи или загрузки данных, содержащими ссылку на логический номер обрабатываемого файла.

Пример 18.18

```
100  DIM A⊙5, B⊙20, C⊙20
. . .
200  DATA LOAD DC OPEN F #2, "АНКЕТА"
. . .
250  DATA LOAD DC #2,  A⊙, B⊙, C⊙, D, E
. . .
300  DBACKSPACE #2, 2S
. . .
400  DATA SAVE DC #2, A⊙, B⊙, C⊙, D, E
. . .
450  DSKIP #2, END
. . .
500  DBACKSPACE #2,  BEG
. . .
600  DATA SAVE DC #2, CLOSE
```

В программе следующего примера создается новый файл «АНКЕТА 2», содержащий анкеты работников из файла «АНКЕТА», поступивших на работу до 1970 г. Для этого последовательно просматриваются записи файла «АНКЕТА» и подходящие выводятся в новый файл «АНКЕТА 2».

Пример 18.19

```
100  DIM A⊙5, B⊙20, C⊙20
110  REM НАЗНАЧЕНИЕ ЛОГИЧЕСКИХ НОМЕРОВ ДИСКОВЫХ УСТРОЙСТВ
120  SELECT #2 18F, #7 18R.
130  REM ОТКРЫТИЕ ФАЙЛОВ
140  DATA LOAD DC OPEN F #2,  "АНКЕТА"
150  DATA LOAD DC OPEN R #7, (100) "АНКЕТА 2"
160  DATA LOAD DC #2, A⊙, B⊙, C⊙, D, E
170  IF END THEN 210
180  IF D> = 1970 THEN 160
190  DATA SAVE DC #7, A⊙, B⊙, C⊙, D, E
200  GOTO 160
210  DATA SAVE DC #7, END
220  DATA SAVE DC CLOSE ALL
```

В строке 160 загружается очередная запись из файла с логическим номером 2 (файл «АНКЕТА»). При попытке загрузить концевую запись этого файла происходит переход к строке 210, в которой записывается концевая запись в файл с логическим номером # 7, т. е. в файл «АНКЕТА 2», В строке 220 стоит оператор закрытия файла с параметром ALL, предназначенным для закрытия всех открытых файлов. В данном примере он заменяет два оператора закрытия

```
DATA  SAVE   DC   #2,  CLOSE
DATA   SAVE   DC   #7,  CLOSE
```

Логический номер в операторах работы с диском может задаваться с помощью числовой переменной. Например, файл «АНКЕТА» можно было бы открыть так:

```
A = 2
DATA   LOAD  DC  OPEN  F#A,  "АНКЕТА"
```

Здесь логический номер задан с помощью числовой переменной А. Еще несколько примеров задания логического номера:

```
DATA  LOAD   DC  OPEN   F#Z1,   (3*H—7),  "D1"
DSKIP  #P,   12
```

18.8.2. ЗАДАНИЕ ТИПА ДИСКА ПАРАМЕТРОМ Т

До сих пор использовались примеры с явным указанием типа диска в операторах открытия файла. Пусть некоторая программа содержит такие операторы с явной ссылкой на тип диска «правый», а диск с файлами данных установлен в левом гнезде дискового устройства (тип диска «левый» — R). Теперь,

чтобы эта программа правильно выполнилась, необходимо во всех операторах открытия файлов исправить тип диска F на R. Модификация программы может быть весьма трудоемка при большом количестве таких операторов и сопряжена с опасностью внесения ошибок в отлаженную программу. Чтобы этого избежать, рекомендуется осуществлять задание типа диска с помощью параметра T. В этом случае тип диска устанавливается таким, каким он был определен в последнем выполненном операторе SELECT с соответствующим номером. Например, операторы

```
100  SELECT  #3  1CR
110  DATA  LOAD  DC  T #3,  "Dl"
```

обеспечивают открытие файла с именем «D1» на съемном диске. Чтобы обеспечить открытие этого файла на другом устройстве, нужно в операторе SELECT изменить адрес и тип диска, например

```
100  SELECT  #3  18F
```

При таком назначении логического номера устройства файл «D1» будет открыт на левом гибком диске. С помощью следующей простой диалоговой процедуры выбора адреса и типа диска можно обеспечить полную независимость программы обработки файлов данных от адресов дисковых устройств.

Пример 18.21

```
100  DIM  P⊙60
110  SELECT  #1  18F,  #2  18R,  #3  1CF,  #4  1CR
120  PRINT  HEX(030A0A0A0A0A0A)
130  PRINT  "18F-1,  1CR-3"
140  PRINT  "18R-2,  1CR-4"
150  GOSUB  '200("ВВЕДИТЕ КОД УСТРОЙСТВА ДЛЯ
ФАЙЛА" ,4 ,1)
160  REM  УСТАНОВКА ЛОГИЧЕСКОГО НОМЕРА ФАЙЛА
170  Z=X
180  DATA  LOAD  DC  OPEN  T  #Z,  "АНКЕТА"
. . .
1000  REM ПОДПРОГРАММА ВВОДА ЧИСЛОВОГО ЗНАЧЕНИЯ
1010  DEFFN  '2OO(P⊙,  Ul,  U2)
1020  PRINT  HEX (010A);  P⊙
1030  INPUT  X:X  =INT(X)
1040  IF  X>U1  THEN  1080
1050  IF  X<U2  THEN  1080
1060  RETURN
1070  PRINT  HEX(0C); "       "; "НЕВЕРНЫЙ ВВОД"
1080  GOTO  1020
```

В этом примере в строке 110 обеспечивается запись возможных адресов и типов устройств в строки таблицы устройств с номерами от 1 до 4. Затем на печать выводится перечень этих адресов с их порядковыми номерами. Затем происходит переход на подпрограмму ввода числового значения (строки 1000—1080). В подпрограмму передаются 3 фактических параметра — текст запроса и ограничения сверху и снизу на принимаемое числовое значение. Если это значение выходит за пределы, то печатается сообщение об ошибке. Если значение переменной X введено правильно, то управление передается основной программе. Затем в строке 180 открывается файл «АНКЕТА» с логическим номером # Z.

Существует и другой способ задания адреса и типа диска с использованием параметра T.

Пример 18.22. Выбор дисков для двух файлов возможен при помощи следующей программы:

```
100  DIM  P⊙60
110  GOSUB  '210("ВВЕДИТЕ КОД УСТРОЙСТВА ДЛЯ ФАЙЛА
'D1'")
120  IF  X=2  THEN  140
130  SELECT  #1  18R:  GOTO  150
140  SELECT  #1  18F
150  REM  "ОТКРЫТИЕ ФАЙЛА  "Dl"
160  DATA LOAD DC OPEN  T  #1,  "D1"
170  GOSUB  '210("ВВЕДИТЕ КОД УСТРОЙСТВА ДЛЯ ФАЙЛА
'D2')"
180  IF  X=2  THEN 200
190  SELECT  #2  18R: GOTO  210
```

```
200  SELECT  #2  18F
210  REM  ОТКРЫТИЕ ФАЙЛА "D2"
220  DATA  LOAD  DC  OPEN  T #   2,   "D2"
. . .
500  DEFFN '  210(P⊙)
510  PRINT  HEX(030A0A0A0A0A0A)
520  PRINT,  "18F-1,  18R-2"
530  GOSUB  '200(P⊙, 2, 1)
540  RETURN
. . .
1000  DEFFN  '200 (P⊙,  U1 ,  U2)
```

В этом примере логический номер устройства и соответствующего файла задан в операторах открытия файла явно. Назначение адреса и типа диска в зависимости от ответа пользователя происходит путем выполнения нужного оператора SELECT. Запрос реализуется в подпрограмме 210. Подпрограмма 200 идентична подпрограмме из примера 18.21.

Использование параметра Т оправдывает себя и в тех случаях, когда нет необходимости в участии пользователя в выборе адреса и типа диска. Пусть разработчику программы кажется, что файл всегда может быть размещен на левом гибком диске. С течением времени может оказаться более удобным держать этот файл на правом диске или на жестком диске. Если при записи программ был использован параметр Т, то при изменении адреса диска бывает достаточно изменить один лишь оператор SELECT без изменения многих операторов работы с файлом.

18.8.3. ОПЕРАТОР LIMITS

Оператор LIMITS позволяет загрузить информацию об адресах каталогизированных файлов в числовые переменные. Им удобно пользоваться для программного контроля числа незанятых секторов файла в различных процедурах ввода данных, а также в программе обработки каталогизированных файлов в режиме абсолютной адресации секторов и т. д. Оператор LIMITS имеет две формы.

Ф о р м а 1

```
LIMITS  <диск>,          <имя> <переменная 1>,     <переменная 2>,
<логический                    <переменная 3>,     [<переменная 4>]
номер файла >
```

где <диск> — тип и адрес диска.

В результате выполнения оператора указанным числовым переменным присваиваются значения: первой — адрес начального сектора файла, второй — адрес конечного, сектора, третьей — количество фактически использованных секторов, четвертой — кода состояния файла.

Для правильного формирования третьего параметра файл должен содержать концевую запись. Код состояния принимает следующие значения:

0 — если файл с данным именем не существует;
1 — если файл является программным;
2 — если файл является файлом данных;
3 — если файл является вычеркнутым программным файлом;
4 — если файл является вычеркнутым файлом данных.

Оператор LIMITS, употребляемый в первой форме, обеспечивает нахождение имени указанного файла в указателе каталога и перепись адресных параметров файла в строку таблицы устройств с заданным номером, если этот номер указан в операторе LIMITS, и в строку с номером #0, если параметр «логический номер файла» в операторе опущен. Затем адресные параметры из таблицы устройств присваиваются числовым переменным, указанным в операторе LIMITS.

П р и м е р 18.23

```
100  LIMITS  F  "AA",  A1,  A2,  A3
110  LIMITS  R#5  "A2",  X,  Y,  Z
120  LIMITS   T#1,   "АНКЕТА",  A(1),  A(2),   P(3)
```

В строке 100 адресные параметры файла «АА» записываются в переменные Al, А2, А3 через нулевую строку таблицы устройств. В строках 110 и 120 запись адресных параметров происходит через 5-ю и 1-ю строки таблицы устройств.

Если в операторе LIMITS указывается номер, уже приписанный открытому файлу, то после выполнения оператора LIMITS в первой форме информация о старом файле теряется.

Форма 2

```
LIMITS      <логический          <переменная 1>,     <переменная 2>,
              номер файла >      <переменная 3>      [<переменная 4>]
```

По оператору LIMITS во второй форме адресная информация переписывается из таблицы устройств с указанным номером (из нулевой строки, если номер не задан) в числовые переменные. Никакие обращения к диску при этом не происходят.

Пример 18.24

```
100  LIMITS  T#3,  A1,  A2,  A3
110  LIMITS  T#2,  P(1),  P(2),  P(3)
```

Адресная информация из третьей строки таблицы устройств переписывается в переменные A1, A2, A3, а из второй — в элементы числового массива P( ).

При использовании оператора LIMITS во второй форме информация переписывается из таблицы устройств независимо от того, связывалась соответствующая строка с дисковым файлом или нет.

Пример 18.25

```
100   SELECT #7 1CR
110   DATA LOAD DC OPEN T #7, "АНКЕТА".
120   DSK1P #7, 5
130   LIMITS T #7, B1, B2, B3
```

В строке 120 происходит присвоение адресных параметров файла «АНКЕТА» переменным B1, B2, B3.

Пример 18.26. Напечатать сообщение об адресных параметрах файла, причем имя файла необходимо ввести с клавиатуры.

```
100   DIM A⊙8
100   INPUT "ВВЕДИТЕ ИМЯ ФАЙЛА", A⊙
110   LIMITS F  A⊙, A, B, C
120   PRINT HEX(030A0A); "ФАЙЛ"; A⊙: PRINT
130   PRINT "НАЧАЛЬНЫЙ СЕКТОР"; A
140   PRINT "КОНЕЧНЫЙ СЕКТОР"; B
150   PRINT "ИСПОЛЬЗОВАНО СЕКТОРОВ";
160   PRINT "ДЛЯ ХРАНЕНИЯ ДАННЫХ"; C - 2
170   GOTO 100
```

В строке 160 напечатано количество использованных секторов за вычетом двух секторов, занятых концевой и служебной записями.

18.9. Режим абсолютной адресации секторов

Режим абсолютной адресации секторов включает ряд операторов, которые позволяют записывать информацию в заданные секторы диска или загружать информацию из них. В этом режиме программные файлы и файлы данных идентифицируются только адресом начального сектора. Для записи или считывания записей из файла в соответствующих операторах должен указываться адрес начального сектора записи. При работе в режиме абсолютной адресации секторов вся ответственность за поддержание адресной информации о файлах ложится на пользователя. Так как операторы этого режима обеспечивают непосредственный доступ к любому сектору, их называют еще операторами прямого доступа.

Операторы режима абсолютной адресации секторов делятся на две группы: операторы DA и операторы BA. Операторы первой группы используются для записи (загрузки) программ и данных, начиная с сектора, адрес которого указывается непосредственно в операторе. В режиме DA информация (программы или данные) записывается в тех же форматах, что и в режиме каталога. Операторы второй группы предназначены для записи или загрузки данных блоками по 256 байт без управляющей информации.

Несмотря на то что в режиме DA нет понятия каталога диска, таблица устройств все же используется для хранения характерной для режима DA адресной информации: адреса начального сектора, указанного в операторе, максимально возможного адреса сектора (зависит от дискового устройства), адреса сектора, следующего за последним использованным сектором при записи или за последним

прочитанным сектором при чтении. Если в операторах работы с диском в режиме абсолютной адресации указывается логический номер (#0 — #7), то упомянутая адресная информация записывается в строку таблицы устройств с соответствующим номером.

Помимо операторов записи и загрузки информации с диска в режиме абсолютной адресации есть операторы для копирования и проверки информации, записанной на диск.

Ниже приводится перечень операторов режима абсолютной адресации секторов и их функций:

SAVE DA	— записывает на диск программу;
LOAD DA	— загружает с диска программу в оперативную память;
DATA SAVE DA	— записывает данные из оперативной памяти на диск в виде одной логической записи;
DATA SAVE DA END	— записывает специальную запись — признак конца данных;
DATA LOAD DA	— загружает данные одной записи и связывает эти данные с переменными в оперативной памяти;
DATA SAVE BA	— записывает блок данных длиной 256 байт из оперативной памяти на диск;
DATA LOAD BA	— загружает с диска в оперативную память блок данных длиной 256 байт;
COPY TO	— копирует диск или часть диска на другой диск;
VERIFY	— проверяет правильность записи информации на диск.

18.9.1. ЗАДАНИЕ АДРЕСА СЕКТОРА

Как отмечалось выше, все операторы режима абсолютной адресации требуют задания адреса начального сектора, начиная с которого записывается или загружается информация. Этот адрес можно задавать в виде выражения либо символьной переменной. В последнем случае в качестве адреса используется двоичное значение двух первых байтов символьной переменной. Значение выражения или символьной переменной, задающее адрес начального сектора, не должно превосходить адреса последнего сектора диска. Выражение или переменная, задающие начальный адрес сектора, записываются в круглых скобках после названия оператора и адреса диска.

После выполнения оператора машина запоминает адрес сектора, следующего за последним сектором, использованным в данной операции. Значение этого адреса можно считать в числовую или символьную переменную, которая может быть указана после адреса начального сектора. В случае символьной переменной используется двоичное значение первых двух байтов. В качестве примера приведем оператор записи программы

```
SAVE DA  F (20)
```

Здесь 20 — адрес сектора, начиная с которого программа, находящаяся в памяти, будет записана на диск. Чтобы узнать, сколько секторов занимает программа (или каков адрес первого сектора, не занятого программой), этот оператор нужно было бы записать так:

```
SAVE DA F (20, A)
```

В переменную A будет записан адрес сектора, следующего за последним сектором, использованным при записи программы.

18.9.2. ЗАПИСЬ И ЗАГРУЗКА ПРОГРАММ

Запись программ на диск в режиме DA осуществляется с помощью оператора SAVE DA. Программа, находящаяся в оперативной памяти, записывается в последовательные сектора диска, начиная с сектора, адрес которого указан в операторе SAVE DA.

Пример 18.27

```
SAVE  DA  F   (50,   C)
SAVE  DA  R   (1+X*(K-1)),   A)
SAVE DA F   (50,   B⊙)
SAVE  DA  F   #3,   (P/5,   A⊙(3))
```

По окончании записи переменной, стоящей в круглых скобках после адреса начального сектора, присваивается значение адреса сектора, следующего за последним занятым. Пусть программа занимает на диске 10 секторов. Тогда значение числовой переменной C будет равно 60, а в первых двух байтах

символьной переменной B⊙ будет записан код HEX (003C).

Программы, записанные по оператору SAVE DA, обычно загружаются с помощью оператора LOAD DA в режиме непосредственного счета. При этом в нем указывается адрес начального сектора, начиная с которого записана нужная программа, например

```
LOAD  DA  F(100)
LOAD  DA   R (100, A)
```

С помощью оператора LOAD DA можно загрузить программу, записанную на диск в режиме автоматической каталогизации. Для этого нужно выполнить оператор LIST DC или LIMITS и запомнить начальный адрес программы. Затем нужно выполнить оператор LOAD DA, указав в нем этот адрес. Отметим, что таким образом можно загрузить вычеркнутую программу.

18.9.3. ЗАПИСЬ И ЧТЕНИЕ ДАННЫХ

В режиме абсолютной адресации данные записываются в виде логических записей, создаваемых оператором DATA SAVE DA. Данные, указанные в этом операторе, записываются на диск, начиная с сектора, адрес которого указан в операторе. Чтобы записать массив A(), начиная с 100-го сектора, нужно выполнить оператор DATA SAVE DA F (100, A) A( ).

Адрес первого сектора, не занятого для хранения элементов массива A(), будет присвоен переменной A. Если записи должны располагаться на диске последовательно, одна за другой, удобно использовать одну и ту же переменную для задания адреса начального сектора записи и запоминания адреса первого незанятого этой записью сектора.

Пример 18.28

```
100   DIM  A(15)
110   P=50  :  REM НАЧАЛЬНЫЙ СЕКТОР 1-Й ЗАПИСИ
120   FOR K=l   TO   15:   PRINT K;
130   INPUT   "ВВЕДИТЕ ЗНАЧЕНИЕ ЭЛЕМЕНТА ЗАПИСИ", A(K)
140   NEXT   K
150   DATA  SAVE  DA  F  (P, P)  A( )
. . .
200   GOTO   120
```

Адрес начального сектора сначала равен 50. После ввода элементов записи (строка 120—140) запись выводится на диск (строка 150), а переменной P присваивается значение 51, так как запись занимает один сектор. Следующая запись будет записана в сектор 51 и т. д. Так же как и в режиме каталога, по окончании записи данных с помощью оператора DATA SAVE DA END можно записать признак конца данных. Наличие такой записи — удобное средство для проверки конца данных во время загрузки данных.

Пример 18.29

```
100   DIM   A(15)
110   P = 50
120   FOR  K=1  TO  15:PRINT K;
130   INPUT   A (K)
140   NEXT  K
150   DATA  SAVE  DA  F  (P, P)  A( )
160   INPUT  "ЕСТЬ ЕЩЕ ЗАПИСИ - 1,  НЕТ -2",   P
170   IF  P = 1 .  THEN  120
. . .
200   DATA  SAVE  DA  F  (P, P)  END
```

Для загрузки записей используется оператор DATA LOAD DA. Данные загружаются, начиная с сектора, адрес которого указан в операторе.

Например, оператор

```
DATA  LOAD   DA  F(100, A)   A, B, C( )
```

загружает значения, записанные оператором DATA SAVE DA, начиная с сектора с адресом 100, и присваивает их переменным, указанным в операторе. Чтобы при загрузке записей можно было обнаружить концевую запись, удобно пользоваться оператором IF END THEN сразу после оператора DATA LOAD DA.

Пример 18.30

```
100  DIM  B(15)
110  P = 50
120  DATA  LOAD  DA  F  (P, P)  B ( )
130  IF  END  THEN  300
. . .
200  MAT  PRINT  B
210  GOTO  120
. . .
300  STOP  "КОНЕЦ ДАННЫХ"
```

При попытке загрузить концевую запись (строка 130) происходит переход на строку 300. Если запись не является концевой, B() печатается в строке 200, загружается следующая запись и т. д.

18.9.4. ОПЕРАТОРЫ BA

Операторы этой группы включают два оператора — DATA SAVE BA и DATA LOAD BA — и используются для записи и загрузки с диска содержимого одного сектора размером в 256 байт. По оператору DATA SAVE BA в заданный сектор записывается содержимое символьного массива. Если массив содержит больше 256 байт, то записываются первые 256. Если массив содержит меньше 256 байт, то оставшиеся байты сектора заполняются кодами HEX (00), например

```
DATA  SAVE  BA  F(100)  A⊙( )
```

Содержимое любого сектора можно загрузить с помощью оператора DATA LOAD BA.

```
DATA  LOAD  BA  F(500)  B⊙( )
```

П р и м е р 18.31. Требуется напечатать содержимое заданного сектора в шестнадцатеричном виде.

```
10  DIM  A⊙(16)
20  INPUT  "ВВЕДИТЕ АДРЕС НУЖНОГО СЕКТОРА",  P
30  DATA  LOAD  WA  F  (P, P)  A⊙( )
40  HEX  PRINT  A⊙( )
50  GOTO  20
```

Заметим, что попытка напечатать массив A⊙() с помощью оператора PRINT могла быть неудачной, если среди загруженных 256 байт были байты, содержащие коды HEX(03), HEX(01) и т. п. Чтобы гарантировать вывод на печать символов из массива A⊙ (), нужно в примере 18.31 заменить строку 40 на следующие:

```
40  FOR  K=l  TO  256
42  IF  STR  (A⊙( ),  K,  1) < HEX(20)  THEN  46
44  IF  STR  (A⊙( ),  K,  1) > HEX( 79) THEN  46:  GOTO  48
46  STR  (A⊙( ),  K,  1) =  "*"
48   NEXT  K
49   PRINT  A⊙( )
```

В этом примере все непечатаемые символы из A⊙() заменяются на символ *. В строке 49 массив A⊙() печатается с помощью оператора PRINT.

18.9.5. ПРОВЕРКА ДИСКОВ

Оператор VERIFY предназначен для контроля правильности записи информации в заданной области диска. Границы области задаются в операторе VERIFY в виде выражений. Например, VERIFY F (250, 400) обеспечивает проверку секторов правого диска с 250-го по 400-й включительно.
Пусть переменная A равна 20. Оператор

```
VERIFY  R (A, A+100)
```

проверяет секторы с 20-го по 120-й. Если границы проверяемой области не заданы, то проверяются все секторы диска. Значение адреса начального сектора должно быть меньше адреса конечного сектора, иначе выдается сообщение об ошибке. Если во время проверки обнаруживаются ошибки, то на экран дисплея выводится сообщение о номере ошибочного сектора, например

```
ERROR  IN  SECTOR 200
```

18.9.6. КОПИРОВАНИЕ ДИСКОВ

Оператор COPY TO предназначен для копирования части или всего диска на другой диск. Границы копируемой области (адреса граничных секторов) могут задаваться в виде выражений. Если эти адреса не указаны, то копируется весь диск полностью, например:

COPY F TO R — копирование правого диска на левый;

COPY (100, 300) TO R (50)—копирование 201-го сектора правого диска на левый, причем содержимое 100-го сектора левого диска копируется в 50-й правого, 101-го — в 51-й и т.д.

В отличие от оператора MOVE, оператор COPY не удаляет при копировании вычеркнутые файлы.

Рекомендуется скопированный диск проверить с помощью оператора VERIFY.

В заключение приведем пример использования операторов режима абсолютной адресации секторов для обработки каталогизированных файлов. В качестве такого файла возьмем файл с именем «АНКЕТА», содержащий анкеты о работниках некоторого предприятия. Будем считать, что записи (анкеты) в этом файле располагаются в порядке возрастания значений поля «учетный номер», которое будем называть ключом записи, и каждая запись занимает на диске один сектор. Необходимо вывести на печать содержимое анкеты с учетным номером, заданным пользователем с клавиатуры. Задачу эту можно решить, пользуясь лишь операторами режима каталога. Для этого нужно открыть файл и последовательно загружать в оперативную память записи до тех пор, пока не встретится запись (анкета) с нужным учетным номером.

Более быстрым и эффективным способом поиска записи с нужным ключом является так называемый двоичный поиск, который заключается в следующем:

1. Заданный ключ сравнивается с ключами первой и последней записей в файле. Эти записи могут быть загружены с помощью оператора DATA LOAD DA. Адреса начального и последнего занятого данными сектора могут быть получены с помощью оператора LIMITS. Если заданный ключ совпадает с ключом первой или последней записи, то задача решена.

2. Заданный ключ не совпадает с ключом первой или последней записи. В этом случае вычисляется адрес сектора, лежащего посередине файла

M=INT((L+H)/2),

где

L — адрес начального сектора файла, H — адрес последнего занятого данными сектора, M — адрес сектора, лежащего посередине файла.

3. Заданный ключ сравнивается с ключом записи из сектора с адресом M. Если заданный ключ меньше ключа записи с адресом M, следует перейти к поиску нужной записи в первой части файла: если больше, то поиск осуществляется во второй части файла.

Для каждой из полученных частей файла адреса начального и конечного секторов известны. Для первой части это 1 и M; для второй — M + 1 и H. Для поиска требуемой записи нужно перейти к пункту 1.

Для применения описанного способа поиска необходимо выполнение трех следующих требований:

записи файла должны быть отсортированы по тому ключу, по которому осуществляется поиск;

все записи файла должны занимать одно и то же число секторов;

должны быть известны адреса начального и конечного секторов файла.

Адресные параметры файла можно получить с помощью оператора LIMITS:

```
LIMITS F "АНКЕТА", A, B, C
```

В переменной A будет находиться адрес начального сектора файла «АНКЕТА». Чтобы получить адрес последнего сектора, использованного для записи данных, нужно сначала получить адрес концевой записи T:

```
T=A+(C—2)
```

Адрес предыдущего сектора, занятого под данные, H:

```
H=T—1
```

Этот адрес используется на первом шаге двоичного поиска. Ниже приводится программа, реализующая описанный алгоритм.

П р и м е р 18.32

```
10 REM ПРОГРАММА ДВОИЧНОГО ПОИСКА
20 DIM A⊙5, B⊙20, C⊙20, K⊙5
30 LIMITS F "АНКЕТА", A, B, C
```

```
40  REM ВЫЧИСЛЕНИЕ АДРЕСА ПОСЛЕДНЕЙ  ЗАПИСИ ДАННЫХ
50  T=A+C-2 .
60  H=T-1
70  REM  ВВОД КЛЮЧА ДЛЯ ПОИСКА
80  INPUT "ВВЕДИТЕ КЛЮЧ НУЖНОЙ  ЗАПИСИ",  K
90  REM ЗАГРУЗКА И ПРОВЕРКА ПЕРВОЙ ЗАПИСИ
100  DATA LOAD DA F (A, S), A⊙, B⊙, C⊙, D, E
110  IF A⊙=K⊙ THEN 290
120  REM ЗАГРУЗКА И ПРОВЕРКА ПОСЛЕДНЕЙ ЗАПИСИ
130  DATA LOAD DA F (H, S) A⊙, B⊙, C⊙, D, E
140  IF A⊙=K⊙ THEN 290
150  REM РАСЧЕТ АДРЕСА СРЕДНЕГО СЕКТОРА
160  M=INT((A+H)/2)
170  REM ЗАГРУЗКА И ПРОВЕРКА СРЕДНЕЙ ЗАПИСИ
180  DATA LOAD DA F (M, S) A⊙, B⊙, C⊙, D, E
190  IF A⊙=K⊙ THEN 290
200 REM K⊙ > или < A⊙ ?
210  IF K⊙ < A⊙ THEN 240
220  A=M
230  GOTO 260
240  H=M
250  REM ВСЕ ЗАПИСИ ПРОВЕРЕНЫ?
260  IF H=M+1 THEN 320
270  GOTO 160
280  REM ПЕЧАТЬ НАЙДЕННОЙ ЗАПИСИ
290  PRINT A⊙, B⊙, C⊙, D, E
300  END
310  REM ЗАПИСЬ НЕ НАЙДЕНА - ВЫДАЧА СООБЩЕНИЯ
320  PRINT  "ЗАПИСЬ НЕ НАЙДЕНА"
330  END
```

В результате выполнения оператора LIMITS переменной A будет присвоено значение начального адреса файла «АНКЕТА», переменной C — число использованных секторов. В строках 50— 60 вычисляется адрес последней записи файла «АНКЕТА». В строке 80 вводится ключ нужной записи. В строках 100—140 загружаются первая и последняя записи файла и их ключи сравниваются с заданным ключом. Если ключи совпадают, то выполняется переход на печать записи (строка 280). Если ключи не совпадают, то заданный ключ сравнивается с ключом записи, расположенной в середине файла. Если в свою очередь они также не совпадают, то определяется, в какой части файла может находиться запись с заданным ключом. В зависимости от этого рассчитываются адреса новых граничных секторов и процесс повторяется (со строки 160) до тех пор, пока будет или найдена нужная запись, или установлено, что записи с таким ключом в файле нет.

Глава 19. СЕГМЕНТАЦИЯ ПРОГРАММ

19.1. Организация программных сегментов

При разработке программ часто возникает ситуация, когда та или иная программа не может быть выполнена из-за недостатка имеющейся оперативной памяти. Напомним, что в оперативной памяти содержатся как текст программы, так и переменные, которые используются программой во время выполнения. Решение этой проблемы естественным образом приводит к мысли разбить большую программу на ряд логически законченных небольших программ, которые будем называть программными сегментами. Каждый из этих сегментов может быть целиком размещен в оперативной памяти вместе с переменными, которые он использует, и затем выполнен.

При выполнении программы, состоящей из нескольких программных сегментов, необходима загрузка и выполнение таких сегментов в нужной очередности. Пусть, например, программа состоит из трех сегментов. Выполнение этой цепочки сегментов можно представить в виде следующей последовательности шагов. Сначала нужно загрузить первый сегмент. По окончании работы этого сегмента необходимо очистить ту часть памяти, которую занимали текст программы первого сегмента и переменные, не нужные для работы второго сегмента. Затем в ОЗУ должен быть загружен и выполнен

второй сегмент и т. д. Такую связь между сегментами можно организовать с помощью операторов LOAD DC (или LOAD DA) в тексте программы. Оператором LOAD DC должно заканчиваться выполнение программы первого сегмента. По его окончании загружается следующий сегмент, который должен содержать оператор LOAD DC, обеспечивающий загрузку третьего сегмента. Каждый раз, когда загружается новый сегмент, часть или весь предыдущий сегмент стирается, чтобы загружаемый сегмент мог разместиться в памяти. Для реализации описанного режима выполнения программ, состоящих из нескольких сегментов, используются следующие четыре оператора:

COM — объявляет переменные или массивы, содержащие данные, общими для нескольких программных сегментов. Указанные в операторе переменные не стираются при загрузке нового сегмента;

LOAD DC — используется в режиме каталога;
стирает все переменные, не помеченные оператором COM, а также часть (или все) программных строк того сегмента, который располагается в памяти в данный момент; загружает указанный сегмент с диска;

LOAD DA — используется в режиме абсолютной адресации секторов; стирает все переменные, не помеченные оператором COM, а также стирает часть (или все) программных строк того сегмента, который располагается в памяти в данный момент; загружает указанный сегмент с диска

COM CLEAR — изменяет тип данных (общие на необщие или необщие на общие).

Перед загрузкой очередного сегмента машина автоматически выполняет очистку памяти от переменных, аналогичную выполнению оператора CLEAR N. Параметр N означает необщие переменные. Оператор CLEAR N стирает все переменные, не объявленные с помощью оператора COM, и оставляет общие переменные без изменения.

Оператор LOAD DC используется в программе для загрузки нового программного сегмента. При выполнении этого оператора имя загружаемого сегмента отыскивается в указателе каталога на указанном диске. Следует различать использование оператора LOAD DC (LOAD DA) в режиме непосредственного счета и в программном режиме, В режиме непосредственного счета оператор LOAD DC (LOAD DA) только загружает программу в оперативную память без ее предварительной очистки.

Общая форма:

LOAD DC <тип диска> <устройство>, <имя>
[<номер строки 1>] [,[<номер строки 2>]] [,<номер строки 3>]

где

<имя>— имя программного файла, подлежащего загрузке с помощью оператора LOAD DC; имя может быть задано символьной константой в кавычках или символьной переменной;

<номер строки 1>—номер строки программы, начиная с которой строки программы, находящейся в памяти, стираются перед загрузкой нового сегмента;

<номер строки 2>—номер последней подлежащей стиранию строки программы, находящейся в оперативной памяти, перед загрузкой нового сегмента;

<номер строки 3>—номер строки программы, начиная с которого должна выполняться программа по окончании операции загрузки нового сегмента.

Оператор LOAD DC должен быть последним исполняемым оператором в строке.

При выполнении оператора LOAD DC машина совершает следующие действия:

выполнение программы прекращается;

выполняется команда удаления строк текущей программы CLEAR P, начиная с номера строки, указанного в параметре <номер строки 1>, и кончая строкой с номером, равным параметру <номер строки 2>. Если параметры <номер строки 1> и <номер строки 2> не указаны, то стирается вся программа. Если указан лишь параметр <номер строки 1>, стирается часть программы, начиная с этого номера. Если задан только параметр <номер строки 2>, стирается часть программы, начиная со строки с наименьшим номером до строки с указанным номером;

выполняется команда CLEAR N для очистки всех необходимых переменных и массивов, а также вся информация о циклах FOR-NEXT и адресах возвратов из подпрограмм;

загружается программный сегмент с указанным именем. Загруженная программа начинает исполняться, начиная со строки с номером, указанным в параметре <номер строки 3>. Если последний не задан, программа начинает выполняться со строки с номером, заданным параметром <номер строки 1>. Если этот параметр не задан или строки с таким номером нет, то программа начинает выполняться со строки с наименьшим номером. Оператор LOAD DC может находиться в той части программы, которая подлежит удалению. Например, при выполнении оператора

```
100 LOAD DC R "ПРОГ 100
```

стирается программный текст, начиная со строки с номером 100, стираются все необщие переменные, и программа с именем «ПРОГ» загружается с диска. Выполнение программы начинается со строки с номером 100. Если такой строки нет, то программа будет выполняться, начиная со строки с наименьшим номером.

При выполнении оператора

```
250 LOAD DG F#3, "ЧАСТЬ 2" 900, 1600, .1200
```

стирается программный текст, начиная со строки с номером 900 до строки с номером 1600 включительно, стираются все необщие переменные, и программа (сегмент программы) с именем «ЧАСТЬ 2» загружается с диска, установленного на устройстве с логическим номером #3. Программа будет выполняться, начиная со строки 1200.

При выполнении оператора

```
200 LOAD DC F #X, A⊙ 500, , 120
```

стирается программный текст, начиная со строки с номером 500 до конца программы, стираются все необщие переменные и загружается сегмент с именем, заданным символьной переменной АО. После загрузки программа начнет выполняться со строки 120.

Если в загружаемом сегменте есть строки с номерами, совпадающими с номерами строк, находящимися в памяти программы, то строки последней замещаются соответствующими строками загружаемого сегмента.

В заключение приведем еще несколько примеров правильного употребления оператора LOAD DC:

```
100 LOAD DC F B⊙, 100, 80
100 LOAD DC R B⊙(3) , , 100
```

Оператор LOAD DA используется в программе для загрузки нового программного сегмента в режиме абсолютной адресации.

Общая форма:

```
LOAD DA <тип диска> <устройство>, (<адрес> [,<переменная>])
[<номер строки 1>] [,[<номер строки 2>]] [<номер строки 3>]
```

где

<адрес>— выражение или переменная, значение которых задает адрес начального сектора загружаемой программы;

<переменная>— общая переменная, в которую записывается адрес первого сектора, не занятого загружаемой программой.

Использование этого оператора аналогично использованию оператора LOAD DC. Ниже дается несколько примеров правильного употребления оператора LOAD DA.

```
100 LOAD DA R (140,A) 200, 300
100 LOAD DA F (3*X+C, B⊙) 100
100 LOAD DA R#5, (200) 200, 300
100 LOAD DA R/1C, (A⊙, B), 300, 100
```

19.2. Обмен данными между сегментами

Чтобы выполнить программу, не помещающуюся в оперативной памяти, разобьем ее на несколько сегментов, загрузим эти сегменты в нужной последовательности и запустим программу. Во время такого процесса часто возникает необходимость в передаче различной информации от одного сегмента к другому. Передача может осуществляться двумя способами. Во-первых, через дисковые файлы, организованные пользователем, во-вторых, путем сохранения значений некоторых переменных и массивов в оперативной памяти во время выполнения операции загрузки очередного сегмента (при выполнении оператора LOAD DC). Так как такие переменные и массивы являются общими для обоих сегментов, то их называют общими переменными. В Бейсике «Искры 226» переменные и массивы могут быть описаны как общие с помощью оператора COM. Если переменные объявлены как общие, то они сохраняются при выполнении оператора LOAD DC, их содержимое и размерности без изменений передаются при загрузке следующего сегмента.

Оператор COM должен стоять в программе до появления первого оператора DIM или первой ссылки на любую необщую переменную.

П р и м е р 19.1. Ниже приведен один из случаев использования оператора COM.

```
10  COM X, Y, A⊙20, C⊙14,D⊙, R01, K⊙(4, 4)8
20  COM R (12), S (14, 14), L⊙(6)2
30  DIM T⊙(14)8, T(14)
40  INPUT "ВВЕДИТЕ ИМЯ ФАЙЛА", F⊙
50  DIM R2(12, 12)
60  INPUT "ВВЕДИТЕ НАЧАЛЬНОЕ ЗНАЧЕНИЕ", R2(1, 1)
```

В этом примере простые числовые переменные X и Y описаны как общие (строка 10). Простые символьные переменные объявляются с указанием их длины (допускается длина символьных переменных от 1 до 253 символов). Если длина символьной переменной не указана (переменная D⊙ строка 10), то она устанавливается машиной, равной 16 по умолчанию. Числовые и символьные массивы объявляются также в операторе DIM. Все операторы COM предшествуют операторам DIM и ссылкам на необщие переменные (строки 30—50).

Оператор DIM должен предшествовать ссылкам на элементы массивов, объявленных в этом операторе (строки 50—60), но он может следовать за ссылкой на другую переменную (F⊙ в строке 40). В отличие от этого оператор COM должен предшествовать всем ссылкам на необщие переменные и всем операторам DIM. Так, если строку 40 перенумеровать в строку 5, то получившуюся программу выполнить нельзя — возникает останов по ошибке.

Общие переменные стираются из памяти лишь в результате выполнения команд CLEAR V или CLEAR. Таким образом, COM — переменные сохраняются при операциях загрузки сегментов LOAD DC и могут использоваться несколькими сегментами. Однако в языке есть оператор COM CLEAR, который позволяет объявить общие переменные как необщие и обеспечить, таким образом, их стирание из памяти при выполнении оператора LOAD DC. Оператор COM CLEAR не стирает переменные из памяти. Чтобы понять смысл использования оператора COM, рассмотрим пример простой программы, использующей общие и необщие переменные, а так же схему размещения переменных в памяти ЭВМ.

Пример 19.2

```
10  COM X, A⊙, R2(6), N⊙(4)30
20  INPUT  K⊙
30  DIM  J (4, 4)
40  FOR K=1  TO  4
50  J(K, K) = 1
60  NEXT  K
```

Схема размещения переменных в оперативной памяти

X	A⊙	R2( )	N⊙( )	K⊙	J( )	K

↑ — граница общих переменных

Все переменные, расположенные левее указателя границы общих переменных, общие. Все переменные, расположенные правее, необщие и стираются при выполнении оператора LOAD DC (LOAD DA)

Так как X является первой переменной, появляющейся в программе (в операторе COM в строке 10), она первой размещается в памяти. Следующие общие переменные и массивы A⊙, R2() и N⊙() размещаются в памяти последовательно вслед за переменной X. Вследствие того, что общие переменные описываются в программе раньше необщих, все общие переменные занимают непрерывный участок памяти, начиная с начала области памяти для хранения переменных. В приведенном примере K⊙ — первая необщая переменная и все последующие переменные, появляющиеся в программе, также необщие.

Общая форма оператора COM CLEAR:

$$\text{COM CLEAR}\left[\left\{\begin{array}{l}<\text{переменная}>\\<\text{массив}>\end{array}\right\}\right]$$

Если оператор COM CLEAR используется в программе без указания переменной, все общие переменные становятся необщими. Выполнение оператора сводится таким образом к перемещению указателя границы общих переменных в начало области памяти для хранения переменных. Так, если программу примера 19.2 дополнить строкой

```
70  COM  CLEAR
```

то после выполнения строки 70 схема размещения переменных будет выглядеть так:

X	A⊙	R2( )	N⊙( )	K⊙	J( )	K

↑ — граница общих переменных

Переменные, объявленные как общие в строке 10, станут теперь необщими и при выполнении оператора LOAD DC будут стерты из памяти.

Если в операторе COM CLEAR указать одну из общих переменных, то эта переменная и все переменные, объявленные вслед за указанной переменной, становятся необщими.

Предположим, что строка 70 нашего примера выглядит так:

```
70  COM  CLEAR   R2( )
```

Тогда после выполнения этой строки схема размещения переменных будет выглядеть следующим образом:

X	A⊙	R2( )	N⊙( )	K⊙	J( )	K

↑ — граница общих переменных

Массивы R2() и N⊙() будут теперь необщими. Массив становится необщим, потому что он объявлен в программе после массива R2() (строка 10). Из этого примера видно, что невозможно сделать массив R2() необщим, оставив массив N⊙() общим. Это можно было бы сделать в том случае, если бы R2 () и N⊙() были первоначально определены таким образом:

```
10   COM  X,  A⊙, N⊙(4)30,  R2(6)
```

т. е. R2() после N⊙(). Тогда при выполнении оператора COM CLEAR R2() массив N⊙() оставался бы общим.

Из всего сказанного ясно, что при использовании оператора COM CLEAR последовательность переменных и массивов в операторах COM имеет существенное значение.

Аналогичным образом оператор COM CLEAR может использоваться для превращения необщих переменных в общие. Так, если строку 70 записать в виде

```
70   COM  CLEAR  K
```

тогда после выполнения этой строки схема размещения переменных была бы следующей:

X	A⊙	R2( )	N⊙( )	K⊙	J( )	K

↑ — граница общих переменных

После выполнения оператора COM CLEAR K массив J() и переменная K⊙ будут включены в общую область памяти, потому что они были определены в программе раньше, чем переменная K.

Вновь загружаемые программные сегменты могут добавлять общие переменные к группе тех общих переменных, которые были переданы предшествующим сегментом. Это добавление может происходить как за счет операторов COM в загружаемом сегменте, так и за счет выполнения оператора COM CLEAR с указанием необщей переменной. Если в цепочке сегментов используются одни и те же общие переменные, то достаточно объявить их в первом сегменте из этой цепочки.

Приведем пример простой программы, состоящей из двух сегментов.

Пример 19.3

Сегмент 1

```
10   COM A
20   B = 5
30   A=B/10
40   PRINT  A, B
50   LOAD  DC  R  "ЧАСТЬ   2"
```

Сегмент 2

```
10   PRINT "СЕГМЕНТ 2"
20   PRINT A, B
30   END
```

Будем считать, что второй сегмент записан на диск с именем «ЧАСТЬ 2», а первый сегмент находится в оперативной памяти. В результате выполнения программы на экране будет напечатано

```
   .5                5
СЕГМЕНТ 2
   .5                0
```

Первая строка чисел — результат работы первого сегмента. В ней напечатаны значения общей переменной A и простой переменной B, за которыми выводится результат работы загруженного второго сегмента — значения тех же переменных. При этом видно, что значение общей переменной A сохранилось («передалось») при загрузке и выполнении второго сегмента.

УКАЗАТЕЛЬ ОПЕРАТОРОВ ЯЗЫКА БЕЙСИК*

* Операторы, не рассматриваемые в данной книге, в указателе отсутствуют.

1. Кодовая таблица микроЭВМ «Искра 226» (код КОИ-8)

старшая цифра кода

Код	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
м л а д ш а я ц и ф р а к о д а 0			Пробел	0	@	P	@	p					ю а б ц д е ф г х и й к л м н о	п я р с т у ж в ь ы з ш э щ ч	Ю А Б Ц Д Е Ф Г Х И Й К Л М Н О	П Я Р С Т У Ж В Ь Ы З Ш Э Щ Ч
	1	НЗ	СУ1 !	1	A	Q	a	q								
	2		СУ2 »	2	B	R	b	r								
	3	КТ	#	3	C	S	c	s								
	4		⊙	4	D	T	d	t								
	5		%	5	E	U	e	u	НС							
	6		&	6	F	V	f	v								
	7	ЗВ	′	7	G	W	g	w								
	8	ВШ	(	8	H	X	h	x								
	9	ГТ	)	9	I	Y	i	y								
	A	ПС	*	:	J	Z	j	z								
	B		+	;	K	[	k	[								
	C	ПФ	,	<	L	\	l	\								
	D	ВК		=	M	]	m	]								
	E		.	>	N	/\	n	/\								
	F		/	?	O	ъ	o	ъ								

Условные обозначения кодов управления: НЗ — перевод курсора в левый верхний угол экрана; КТ — очистка экрана и перевод курсора в левый верхний угол экрана; ЗВ — звуковой сигнал дисплея; ВШ — перемещение курсора на экране на один символ влево; ГТ — перемещение курсора на экране на один символ вправо; ПС — перемещение курсора на экране на одну строку вниз; ПФ — перемещение курсора на экране на одну строку вверх; ВК — перемещение курсора на экране в первую позицию текущей строки; НС — перемещение курсора на экране в первую позицию следующей строки; СУ1—переход к высвечиванию на экране в позитивном изображении; СУ2 — переход к высвечиванию на экране в негативном изображении.

2. Физические адреса устройств (ФАУ)

Шестнадцатеричное значение ФАУ	Тип устройства ввода-вывода
01	Устройство клавиатуры
05	Блок отображения символьной информации
0C	Алфавитно-цифровое печатающее устройство
10	Блок отображения графической информации
14	Графопостроитель
18	Сдвоенный накопитель на гибких магнитных дисках
1B	Накопитель на девятидорожечной магнитной ленте
1C	Сдвоенный накопитель на кассетных магнитных дисках
2D	Интерфейс радиальный параллельный (ИРПР)
34	Телекоммуникационный интерфейс (передатчик)
35	Телекоммуникационный интерфейс (приемник)
36	Интерфейс радиальный последовательный (ИРПС передатчик)
37	Интерфейс радиальный последовательный (ИРПС приемник)