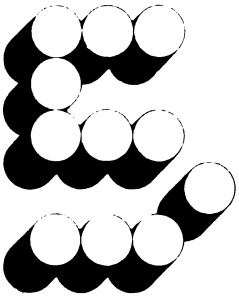
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МИКРОЭВМ

В ОДИННАДЦАТИ КНИГАХ

Под редакцией
лауреата Государственной премии СССР,
доктора технических наук,
профессора
В.Ф.ШАНЬГИНА

КНИГА 

В.Ф.Шаньгин, А.Я.Пьянзин

ДИАЛОГОВЫЙ ЯЗЫК «БЕЙСИК»

МОСКВА «ВЫСШАЯ ШКОЛА» 1987

ББК 32.97
П78
УДК 681.3

РЕКОМЕНДОВАНО ГОСПРОФОБРОМ СССР В КАЧЕСТВЕ ПРАКТИЧЕСКОГО ПОСОБИЯ.

Рецензенты: проф. П. П. Сыпчук (зав. кафедрой вычислительной техники Московского института электронного машиностроения); М. Л. Гуткин (Институт проблем информатики АН СССР); Р. А. Аваков — препод. СПТУ № 127 г. Москвы.

П78 **Программное** обеспечение микроЭВМ: Практ. пособие для инж.-пед. работников системы проф-техн. образования: В 11 кн./Под ред. В. Ф. Шаньгина. Кн. 5./Диалоговый язык БЕЙСИК. В. Ф. Шаньгин, А. Я. Пьянзин. — М.: Высш. шк., 1987. — 111 с.: ил.

В пятой книге пособия описан диалоговый язык БЕЙСИК, рассмотрены различные интерпретирующие системы, позволяющие проводить ввод и редактирование программ, трансляцию и исполнение, средства отладки программ на микроЭВМ.

П $\frac{2405000000-547}{052(01)-87}$ 63—87

ББК 32.97
6Ф7.3

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Одним из основных направлений научно-технического прогресса является компьютеризация, т. е. массовое производство и широкое применение в народном хозяйстве электронных вычислительных машин (ЭВМ).

Благодаря успехам технологии микроэлектроники созданы микропроцессорные вычислительные машины — микроЭВМ (микрокомпьютеры), которые не уступают по вычислительной мощности средним ЭВМ начала 70-х годов. Овладение знаниями и навыками использования современной вычислительной техники является обязательным требованием подготовки учащихся средней общеобразовательной и профессиональной школы. От уровня компьютерной грамотности трудящихся и прежде всего молодого поколения существенно зависят темпы экономического развития страны.

МикроЭВМ (микрокомпьютер), как и другие ЭВМ, имеет следующие основные части: процессор, память, устройства ввода — вывода (рис. В.1).

Процессор является центральным устройством ЭВМ. Он производит арифметические и логические действия над числами и управляет другими устройствами ЭВМ. Для этого в процессоре имеются арифметико-логическое устройство (АЛУ), которое выполняет вычисления, и устройство управления, которое расшифровывает команды и формирует управляющие сигналы. Схема процессора современной микроЭВМ очень сложна, однако в микроэлектронном исполнении она размещается на маленьком кристалле, не превышающем по размерам копеечной монеты.

Память ЭВМ служит для записи, хранения и считывания различной информации: исходных данных, программ, результатов счета. Память состоит из ячеек. *Ячейка памяти* представляет собой устройство для хранения числа или команды (либо части числа или команды). Все ячейки памяти пронумерованы. Номер ячейки служит ее адресом. При записи в память данные получают конкретные адреса (*1, 2, ..., n*). По этим адресам они могут быть впоследствии прочитаны. Запись информации в память приводит к стиранию того, что в ней находилось ранее. При чтении прочитанное сохраняется в памяти неизменным.

В ЭВМ используются три вида памяти — постоянная, оперативная и внешняя. *Постоянная память* служит для хранения неизменной информации, которая может только считываться процессором. Постоянную память иногда называют постоянным запоминающим устройством (ПЗУ). *Оперативная память* дает возможность записать в нее информацию и читать. После выключения ЭВМ содержимое оперативной памяти теряется. Оперативную память иногда называют *оперативным запоминающим устройством* (ОЗУ). *Внешняя память* (накопители на магнитных дисках и магнитных лентах) служит для записи, хранения и чтения информации. В отличие от оперативной памяти она обеспечивает длительное хранение информации больших объемов, однако чтение и запись информации производятся значительно медленнее.

Устройство ввода — вывода персональной микроЭВМ (дисплей с клавиатурой, печатающее устройство) служат для ввода в ЭВМ исходных данных и программ и вывода полученных результатов. Иногда их называют терминалами. Клавиатура микроЭВМ предназначена для ввода цифровой, буквенной и символьной информации. Она имеет много общего с клавиатурой обычной пишущей машинки. *Дисплей* — это устройство отображения информации на экране электронно-лучевой трубки. *Печатающее устройство* предназначено для вывода различной информации на бумагу в виде печатного документа. По принципу работы печатающее устройство похоже на пишущую машинку.

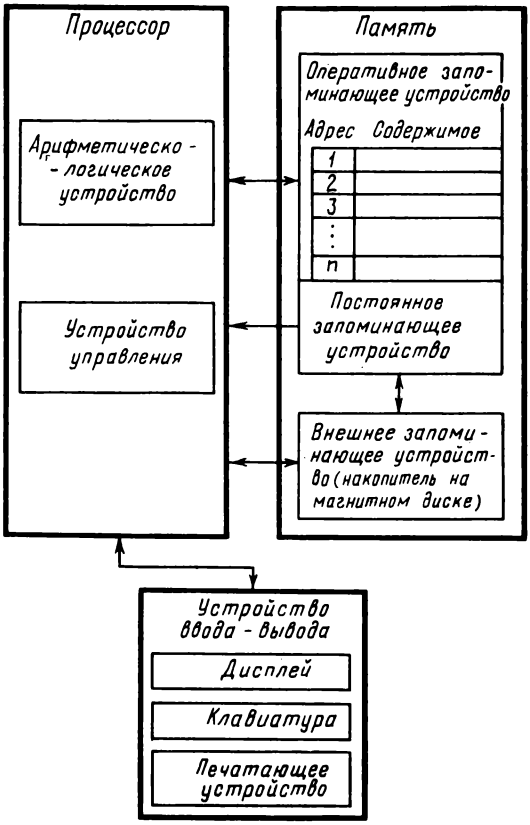


Рис. В.1. Структурная схема микроЭВМ

Первоначально ЭВМ использовались только для вычислений. Однако в дальнейшем они стали успешно выполнять другие функции, связанные с переработкой информации: управлять станками и роботами, распознавать образы и речь, работать в качестве диспетчеров и т. п. Тем не менее ЭВМ не может заменить человека во всех сферах его умственной деятельности. Вычислительная машина может успешно решить задачу, для которой известен алгоритм решения.

Алгоритм — это последовательность операций, однозначно задающая решение определенной задачи обработки данных. Решение задачи на вычислительной машине сводится к выполнению в определенном порядке элементарных действий: сложения, вычитания, умножения и т. д. Для несложных задач вычислительного характера алгоритмы составляются достаточно просто, для задач невычислительного характера — создать хорошие алгоритмы достаточно трудно. Алгоритмы могут быть описаны как словесно, так и изображены графически в виде последовательности условных графических символов (овала, прямоугольника, ромба и т. д.). Графическое изображение алгоритма называется *схемой алгоритма*. Для записи алгоритмов используются специальные алгоритмические языки.

Итак, чтобы ЭВМ смогла решить некоторую конкретную задачу, ей необходимо задать алгоритмы решения этой задачи, причем алгоритм надо записать на языке, понятном машине.

Введем некоторые понятия.

Команда определяет, какое действие и над какими числами должна выполнить машина. Кроме того, в ней указывается, где взять исходные числа и куда отправить результат. Команда, зашифрованная последовательностью единиц и нулей, которые определяют все необходимые действия и адреса ячеек, называется *машинной командой*.

Программа представляет собой последовательность команд, которые должна выполнить машина при решении задачи. Иначе говоря, программа — это алгоритм, записанный на языке, понятном вычислительной машине. Вычислительная машина понимает язык только машинных команд.

В машинном языке каждой команде соответствует свой двоичный цифровой код. В процессе работы машины эти команды, хранящиеся в памяти, будут поочередно считываться устройством управления, расшифровываться там и выполняться.

Программа, написанная на машинном языке, получается очень длинной, трудночитаемой и при составлении требует тщательной и весьма трудоемкой работы.

Процесс составления программ называют *программированием*. В настоящее время для упрощения составления программ разработано большое число языков программирования. Понятия «язык программирования» и «алгоритмический язык» часто выступают как синонимы. Являясь доступными человеку и похожими на привычный математический язык формул, они в то же время могут быть понятны ЭВМ. Программа, написанная на языке программирования, представляет собой последовательность операторов, т. е. укрупненных предписаний ЭВМ по выполнению действий. При этом сама ЭВМ занимается переводом программы с языка программирования на машинный язык. Этот перевод осуществляется с помощью специальных программ — трансляторов или интерпретаторов.

Решение задач с помощью ЭВМ включает в себя следующие основные этапы: постановку задачи, выбор алгоритма, программирование, отладку программы и решение. Эти этапы подробно описаны в кн. 3 и 11 данной серии учебных пособий.

Одним из наиболее распространенных языков программирования является язык БЕЙСИК, разработанный в 1965 г. для составления программы решения задач вычислительного характера с небольшим объемом исходной информации в режиме диалога «человек — машина». Он является ведущим диалоговым языком, используемым практически во всех персональных ЭВМ.

Характерной чертой для режима диалога является то, что пользователь сразу же получает ответы на свои сообщения, посланные в ЭВМ. Название языка возникло из начальных букв английских слов Beginner's All-purpose Symbolic Instruction Code, в переводе означающих «многоцелевой язык символических инструкций для начинающих» (BASIC).

В нашей стране язык БЕЙСИК реализован для таких ЭВМ, как БЭСМ-6, ЕС ЭВМ, М-7000, М-6000, СМ ЭВМ, для ряда мини- и микроЭВМ «Электроника-60», «Электроника-НЦ», «Диалоговый вычислительный комплекс ДВК» и др. Распространению этого языка способствовала простота реализации, а также близость его к широко используемому в нашей стране языку ФОРТРАН. Программа, составленная на алгоритмическом языке БЕЙСИК (исходная программа), преобразуется в язык команд конкретной ЭВМ (рабочую программу) с помощью специальной программы, называемой *интерпретирующей системой* БЕЙСИК (БЕЙСИК-системой).

Как правило, БЕЙСИК-системы ориентированы на определенный тип ЭВМ и могут отличаться друг от друга по своим возможностям. БЕЙСИК-система представляет пользователю следующие возможности:

ввод и редактирование исходной программы на языке БЕЙСИК с выдачей сообщений об ошибках;

вывод на дисплей или печатающее устройство текста исходной программы или ее фрагментов;

анализ, проверку и выполнение операторов языка БЕЙСИК. Во время счета по рабочей программе БЕЙСИК-система организует контроль вычислений и вывод сообщений об обнаруженных ошибках;

хранение исходной информации на внешних носителях — магнитных дисках и магнитных лентах.

В дальнейшем не следует путать понятия «язык БЕЙСИК» и «БЕЙСИК-система».

Цель данной книги — изучение основ программирования на языке БЕЙСИК. Авторы не ставили перед собой задачу полного изложения различных версий языка и описания всех возможностей, которые он допускает. Начинающий программист должен в первую очередь сосредоточить свое внимание на сущности программирования.

Авторы выражают благодарность рецензентам — зав. кафедрой вычислительной техники проф. П. П. Сыпчуку (Московский институт электронного машиностроения) и М. Л. Гуткину (Институт проблем информатики АН СССР), замечания которых способствовали улучшению материала учебного пособия.

Глава 1

ОБЩИЕ СВЕДЕНИЯ О ЯЗЫКЕ «БЕЙСИК» И СТРУКТУРЕ ПРОГРАММ

1.1. ОСНОВНЫЕ СИМВОЛЫ ЯЗЫКА БЕЙСИК

На алфавит языка БЕЙСИК влияет его ориентация на диалоговые вычислительные комплексы и системы, в которых пользователь общается с ЭВМ с помощью клавиатуры дисплея. Поэтому алфавит языка может учитывать специфику клавиатуры и версию языка.

Основные символы языка БЕЙСИК, объединенные в группы, представлены ниже.

Прописные латинские буквы

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

Прописные русские буквы

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

Цифры

0 1 2 3 4 5 6 7 8 9

Знаки арифметических операций	**Знаки операций отношения**
+ плюс	= равно
— минус	> больше
* знак умножения	< меньше
/ знак деления	>= больше или равно (не меньше)
∧ знак возведения в степень (иногда ↑ либо ¬)	<= меньше или равно (не больше)
	<> не равно

Разделительные, другие специальные и служебные символы

. точка	# номер
, запятая	¤ знак денежной единицы
; точка с запятой	% процент
: двоеточие	& коммерческое И
␣ пробел (пустая позиция)	_ знак подчеркивания

' апостроф	? вопросительный знак
" кавычки	@ коммерческое «ЭТ»
(круглая скобка левая	\ черта с левым наклоном
) круглая скобка правая	! восклицательный знак
[,] квадратные скобки	

Прописные латинские буквы применяются для обозначения и написания операторов (см. параграф 1.6) и наименований различных величин, используемых в программах; прописные русские буквы — только в строковых константах (см. параграф 4.1) и пояснительных текстах; цифры — для записи чисел, а также в качестве символов для наименований различных объектов программы.

Чтобы не путать букву О и цифру 0, рекомендуется при написании программ перечеркивать косой чертой цифру.

Группы знаков арифметических операций и знаков операций отношений служат соответственно для записи арифметических выражений и операций отношений в программах.

Назначение и использование разделителей, других специальных и служебных символов будет рассматриваться по ходу изложения материала.

1.2. КОНСТАНТЫ

В языке БЕЙСИК используются числовые и строковые константы. Для каждой константы в памяти ЭВМ выделяется одна или несколько ячеек памяти.

Числовая константа — величина, записанная в программе в виде конкретного числа. Различают два типа числовых констант — целые и вещественные.

Целые константы представляют заданное целое число абсолютно точно. Это бывает необходимо в ряде случаев, когда определенная величина принципиально не может быть представлена приближенно. Так, нельзя сказать 6,999999 человек, либо 7,000001 раз повторить вычисление по формуле.

Вещественные константы представляют заданное число с конечной точностью, пусть даже очень высокой. Так, число 7 может быть представлено, например, как 6.99999 либо как 7.00001, либо как 7.0. Это обусловлено способом представления вещественного числа в ЭВМ. Вещественные константы используются в большинстве расчетов.

В языке БЕЙСИК числа записываются не совсем так, как мы к этому привыкли. Константы каждого типа имеют свою специфическую форму записи.

Целая часть вещественного числа отделяется от дробной части не запятой, а десятичной точкой. Целая и дробная части могут отсутствовать. Знак «+» у положительных чисел также может отсутствовать.

Для записи очень больших или очень малых чисел используют представление в виде числа, умноженного на 10 в некоторой степени. Так,

число 2 500 000 можно записать $2{,}5 \cdot 10^6$. В языке БЕЙСИК это число может быть представлено как 2.5E6. Таким образом, вместо часто встречающегося в обычной записи умножения числа m на степень числа 10 применяется символическая запись $m\text{E}\pm p$, означающая умножение вещественного числа m на $10^{\pm p}$. При таком представлении числа величина m называется *мантиссой* числа, а p — *порядком* числа. Для версии языка БЕЙСИК, реализованной в ДВК, m должно находиться в диапазоне от 1 до 10.

Приведем ниже ряд примеров правильной записи вещественных констант:

Число	**Запись на языке БЕЙСИК**
6	6
—4,2	—4.2
2000	2000 (+2000)
0,005	.005
$5{,}6 \cdot 10^4$	5.6E4 (+5.6E+4)
—0,00072	—7.2E—4
$100000 = 1 \cdot 10^5$	1E5

При записи с десятичным порядком число, стоящее после буквы Е, может быть только целым числом.

Для ряда мини- и микроЭВМ диапазон допустимых чисел по абсолютной величине составляет от 10^{-38} до 10^{38}. Использование в программе вещественных констант вне указанного диапазона вызывает сообщение об ошибке.

При записи чисел нужно учитывать, что ЭВМ производит все операции с конечной точностью. Поэтому нецелесообразно задавать машине числа со слишком большим количеством значащих цифр. Допустимое количество значащих цифр в записи числа равно 6.

Целая константа записывается в виде конечной последовательности десятичных цифр со знаком «+» или «—» перед ней, которая оканчивается знаком процента (%). Например, 7%, —125%, 10 000%. Символ % является признаком целой константы.

Диапазон использования целых констант от —32 768% до +32 767%. Операции с целыми константами вне этого диапазона вызывают сообщение об ошибке.

В некоторых версиях языка БЕЙСИК при записи вещественных констант обязательна десятичная точка, а признаком целых констант является отсутствие десятичной точки.

Строковая константа — это заключенная в кавычки последовательность букв, цифр и других символов, входящих в алфавит языка БЕЙСИК. В этой последовательности могут использоваться и русские буквы. Строковые константы не имеют числового значения и характеризуются последовательностью и составом символов. Их используют при решении различных информационных задач и задач по обработке текстовой информации. Например,

«ТАБЛИЦА»
«СИДОРОВ И. П.»
«X=»

Строковые константы используются при обработке текстов, а также для вывода пояснительного текста при выполнении программы и оформления результатов счета. Ее длина может изменяться от 0 (пустая строка) до 255 символов.

1.3. ПЕРЕМЕННЫЕ

Переменная — это величина, значение которой может изменяться в процессе выполнения программы. Для обозначения переменных в языке БЕЙСИК используются имена, называемые *идентификаторами*. Для каждой переменной в памяти ЭВМ выделяется одна или несколько ячеек памяти. Имя переменной служит как бы адресом ячейки, в которой хранится значение переменной. Указав в программе имя, мы можем извлечь из ячейки памяти значение переменной. Имена переменных участвуют в программе на языке БЕЙСИК вместо данных, которые они представляют.

Как и константы, переменные могут быть трех типов: 1) вещественные, 2) целые, 3) строковые.

Вещественная переменная обозначается одной латинской буквой или латинской буквой, за которой следует цифра. Например, A, B5, C1, Z0, Y9. Она может принимать любое значение, допустимое для вещественной константы. В пределах одной программы можно использовать $26+26\cdot10=286$ различных имен вещественных переменных (26 — количество латинских букв; $26\cdot10$ — количество латинских букв с цифрой).

Целая переменная обозначается так же, как и вещественная, но должна заканчиваться знаком процента %. Например, A%, C8%, B2%, I%. Она может принимать любое из значений, допустимых для целой константы.

Строковая переменная обозначается либо буквой, либо буквой с цифрой, за которыми следует знак ¤. Например, B¤, C1¤, A5¤. Она может принимать любое из значений, допустимых для строковой константы.

Переменные, имена которых начинаются с одних и тех же буквенно-цифровых символов, представляют три различных переменных, например:

A5 — вещественная переменная;
A5% — целая переменная;
A5¤ — строковая переменная.

Примеры неверной записи имен:

2C (начинается с цифры); Ю1 (русские буквы запрещены);
CA (второй символ — буква); A205 (слишком длинное имя);
¤2 (начинается с символа денежной единицы);
%K (начинается с символа процента);
B* (содержит символ *).

Кроме рассмотренных переменных БЕЙСИК допускает использование переменных с индексами (см. параграф 3.1).

1.4. СТАНДАРТНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Для вычисления наиболее распространенных элементарных математических функций (например, синуса, косинуса, экспоненты, квадратного корня, логарифма и др.) в языке БЕЙСИК применяют встроенные стандартные функции. Для их обозначения используют, как правило, трехбуквенные имена. Аргумент функции заключается в круглые скобки. Аргументом функции может быть произвольное арифметическое выражение. Для обращения к функции нужно набрать ее имя и указать аргумент.

Стандартные функции, применяемые в языке БЕЙСИК, приведены в табл. 1.1.

Таблица 1.1

Обозначение функции на языке БЕЙСИК	Пояснение
SIN(X)	Функция синуса ($\sin x$). Вычисляет синус аргумента X
COS(X)	Функция косинуса ($\cos x$). Вычисляет косинус аргумента X
ATN(X)	Функция арктангенса ($\operatorname{arctg} x$). Вычисляет арктангенс аргумента X, находящегося в диапазоне от $-\pi/2$ до $+\pi/2$
EXP(X)	Экспоненциальная функция (e^x). Вычисляет E∧X, где E = 2,71828...
LOG(X)	Функция натурального логарифма ($\ln x$). Вычисляет натуральный логарифм положительного аргумента X
LOG10(X)	Функция десятичного логарифма ($\lg x$). Вычисляет десятичный логарифм положительного аргумента X
SQR(X)	Функция квадратного корня ($\sqrt{x}$). Вычисляет квадратный корень положительного аргумента X
ABS(X)	Функция «абсолютная величина» ($\|x\|$). Вычисляет абсолютное значение (модуль) аргумента X
INT(X)	Целочисленная функция. Определяет наибольшее целое число, не превосходящее аргумент X
SGN(X)	Функция знака ($\operatorname{sign} x$) $\mathrm{SGN(X)} = \begin{cases} +1, & \text{если } X>0. \\ 0, & \text{если } X=0. \\ -1, & \text{если } X<0. \end{cases}$
RND(X)	Функция случайных чисел. Выдает случайное число, лежащее в интервале от 0 до 1. Значение аргумента X игнорируется
PI	Выдает значение числа $\pi = 3{,}1415927$. Может использоваться как числовая константа

Аргументы тригонометрических функций SIN и COS должны быть заданы в радианах. Если угол выражен в градусах, то его нужно перевести в радианы по формуле

Значение в радианах = значение в градусах∗PI/180

В некоторых версиях языка БЕЙСИК имеется только одна функция LOG(X) для вычисления натурального логарифма числа X. Однако натуральные логарифмы легко перевести в логарифмы с любым основанием, используя формулу

$$\log_a N = \ln N / \ln a,$$

где a — основание.

Целочисленная функция INT(X) вычисляет значение наибольшего целого числа, не превышающее значения аргумента X. Например, INT(34.67) = 34.

Чтобы выполнить округление до ближайшего целого, достаточно написать INT (X+.5). Например, число 2.9 будет округлено до 3, а число 2.3 до 2:

INT (2.9+.5) = INT (3.4) = 3

INT (2.3 +.5) = INT (2.8) = 2.

Функция INT(X) может использоваться для округления до любого заданного десятичного разряда с помощью следующего выражения:

INT (X∗10∧D) +.5)/10∧D

где D — целое число, задаваемое пользователем и определяющее точность округления (количество знаков после запятой).

Аргумент функции RND не используется и может быть любым числом. Функция работает как генератор случайных чисел, т. е. вырабатывает произвольные числа из диапазона (0,1) с равномерным законом распределения. Использование RND в качестве операнда выражения эквивалентно замене этого операнда очередным случайным числом. Случайные числа можно получать в различных диапазонах. Например, для получения случайных цифр от 0 до 9 можно воспользоваться выражением

INT (10∗RND(0))

Можно получать случайные числа в заданном интервале. Для получения случайных чисел в интервале (A, B) используется выражение

(B—A)∗RND(0)+A

1.5. ВЫРАЖЕНИЯ

Над числовыми константами, переменными и стандартными функциями можно производить обычные арифметические операции.

Арифметическое выражение — это символическая запись, указывающая правило вычисления числового значения. На языке БЕЙСИК арифметическое выражение записывается в форме, близкой к естественной, общепринятой. Оно состоит из чисел (констант), имен переменных, функций, знаков арифметических операций (символов) и скобок.

Символами арифметических операций являются: ∧ (возведение в степень); ∗ (умножение); / (деление); + (сложение); — (вычитание).

При записи выражения необходимо учитывать следующие рекомендации и ограничения:

1. Формулу требуется записывать в строку без каких-либо подстрочных или надстрочных знаков. Примеры записи арифметических выражений:

Математическая запись	Запись на языке БЕЙСИК
$\frac{ax^3+bx+c}{d^2-3,4}$	(A∗X∧3 + B∗X + C)/(D∧2 + 3,4)
$\frac{\sin 2x \cos x}{2\sqrt{b}}$	SIN(2∗X) ∗ COS(X)/(2∗SQR(B))
$p+(\ln z^2)^2$	P + (LOG(Z/2))/2

2. Следует использовать круглые скобки для указания порядка действий, особенно в сомнительных случаях. Вычисления в скобках производятся в первую очередь. Если выражение, содержащее скобки, само заключено в круглые скобки, то вычисления производятся, начиная с внутренних скобок.

Внутри скобок действия выполняются слева направо в соответствии со старшинством (приоритетом) операций: сначала вычисляется значение функций; затем выполняются все операции возведения в степень; затем — умножения и деления и, наконец, сложения и вычитания.

Если математическое выражение содержит несколько арифметических операций одинакового старшинства, то такие операции выполняются последовательно слева направо. Например, выражение на языке БЕЙСИК −2+A/B∗C+3∧2 эквивалентно математическому выражению $-2+AC/B+3^2$.

Алгоритм его вычисления выполняется в такой последовательности: а) X = 3^2; б) Y = (A/B)C; в) Z = −2+Y+X. Запись вида A∧B∧C для ДВК соответствует записи (A ∧ B) ∧ C.

3. Два знака арифметических операций нельзя ставить рядом, а также нельзя опускать знак умножения между сомножителями. Например, математическое выражение $3xy/-z$ на языке БЕЙСИК можно записать так: 3∗X∗Y/(−Z).

4. Выполнение арифметических операций над арифметическими выражениями одного типа дает результат того же типа:

A% + B% + 10% = целое число
CI ∗ D = вещественное число
7%/9% = 0%, т. е. значение правильной дроби

оказывается равным нулю.

5. Операция над целой и вещественной величинами дает вещественный результат:

A∗B% = вещественное число
5.4∗3% = 16.2

6. Возведение в целую степень выполняется многократным умножением:

$$A \wedge 3\% = A*A*A$$
$$C*J\% = \underbrace{C*C*...*C}_{J \text{ раз}}$$

7. Вычисление результата осуществляется с помощью функций EXP и LOG, если показатель степени — действительное число:

$$X \wedge Y = EXP\,(Y*LOG(X))$$

Характерные ошибки при записи арифметических выражений:

Типичные ошибки в записи формул	Пояснение
3A + B	Пропущен знак умножения
2* – B	Два знака операции следуют подряд
SIN X + B	Отсутствуют скобки для аргумента функции

1.6. ПРОГРАММА НА ЯЗЫКЕ БЕЙСИК

Программа на языке БЕЙСИК записывается в виде последовательности пронумерованных строк. В каждой строке может стоять один или несколько операторов.

Общий вид строки:

⟨*номер строки*⟩ — ⟨*оператор 1*⟩\⟨*оператор 2*⟩\...\⟨*оператор* n⟩ ВК

Здесь *номер строки* — целое число в диапазоне от 1 до 9999*; ␣ —пробел (пустая позиция), *оператор* — оператор языка БЕЙСИК; \ — разделитель операторов (иногда : или #); ВК — возврат каретки, означающий конец строки.

Здесь и далее угловые скобки служат для выделения отдельных элементов конструкции языка.

Оператор — это предписание ЭВМ, написанное на языке БЕЙСИК. Он содержит указание машине, что надо выполнить в данный момент. Оператор состоит из специально зарезервированного слова (слов)** и данных.

Общий вид оператора:

⟨*имя*⟩ ⟨*содержание*⟩

---

* В разных версиях БЕЙСИК-системы максимальные значения номера строки могут различаться.

** Слова, имеющие строго фиксированное написание и назначение в языке, называются *зарезервированными словами*. Их следует употреблять не произвольно, а в строгом соответствии с правилами языка

Здесь *имя* — специальное слово, написанное на английском языке, которое имеет строго фиксированное написание и обозначает, что должна выполнить ЭВМ; *содержание* — это данные, необходимые для выполнения оператора.

Например

LET A = B

имя содержание

Кроме имени, в состав оператора могут входить еще некоторые зарезервированные служебные слова. Они предназначены для уточнения действий, выполняемых оператором. Например,

IF A<B THEN 50

имя содержание служебное слово содержание

Некоторые операторы могут содержать только имя, например STOP. Подробно правила написания каждого оператора (его синтаксис) и употребление будут рассмотрены в соответствующих разделах.

Одна строка на языке БЕЙСИК может содержать до 255 символов и таким образом занимать несколько строк на экране. При начальном знакомстве с языком в программах для большей ясности следует располагать один оператор в строке. Рекомендуется нумеровать строки через 10, чтобы при внесении изменений в программу можно было вставлять новые строки с операторами. Пример записи строк-операторов:

```
10 — LET A = S
40 — PRINT C
```

Здесь переменной A присваивается значение 5. Оператор PRINT обеспечивает вывод значения переменной C.

Операторы языка БЕЙСИК делятся на две группы:

1) выполняемые операторы — определяют действия программы, указывая БЕЙСИК-системе, какую операцию нужно выполнить (PRINT — *печатать*, READ — *читать*, GO TO — *идти к* и т. д.);

2) невыполняемые операторы — описывают характер и упорядочение данных, позволяют вводить в программу примечания и сообщения описательного характера (REM — *комментарий*, DIM — *размерность*).

Для управления работой БЕЙСИК-системы следует использовать специальный набор системных команд. *Системная команда* — это указание, с помощью которого пользователь дает приказ БЕЙСИК-системе на выполнение тех или иных действий. Эта команда записывается без номера строки.

На этапе знакомства с языком будем пользоваться тремя системными командами: NEW (*новая*), RUN (*пуск*) и LIST (*вывод текста программы*). Более подробно о системных командах см. в гл. 6.

Если нужно начать ввод программы, то сначала вводят команду NEW. Она может содержать имя вводимой программы. По этой команде ЭВМ стирает ранее введенные программы и готова к вводу новой программы; при этом на экран выводится сообщение о готовно-

сти к работе READY (*готов*). Теперь можно вводить программу, набирая на клавиатуре строки программы, которые автоматически высвечиваются на экране дисплея. Строки можно набирать в любом порядке. В памяти ЭВМ они автоматически располагаются по возрастанию номеров и в этом порядке впоследствии обрабатываются БЕЙСИК-системой. После ввода всей программы можно начинать ее выполнение. Для этого задается команда RUN. Пример записи и выполнения простейшей программы:

READY	Сообщение ЭВМ о готовности к работе
NEW	Команда подготовки ЭВМ к вводу новой программы
READY	Сообщение ЭВМ о готовности к работе
10 LET X = 7 20 LET Y = SQR(X) 30 PRINT X, Y 40 END	Программа пользователя (исходная программа)
RUN	Команда пуска
7 2.64575	Результат счета, выданный машиной
READY	Сообщение ЭВМ о готовности к работе

Следует отметить, что в отличие от операторов языка БЕЙСИК все системные команды записываются без номеров строк.

Операторы программы выполняются в порядке возрастания номеров строк до тех пор, пока не встретится оператор передачи управления, оператор останова или оператор конца текста программы.

Для уплотнения записи программы в большинстве версий языка допускается на одной строке записывать несколько операторов. При этом они разделяются друг от друга либо с помощью черты с левым наклоном (\), либо двоеточием (:), либо знаком номера (≠). Например,

20 LET Y = SQR (X)\PRINT Y

Порядок выполнения операторов в пределах строки естественный — слева направо. Отсутствие номеров у внутренних операторов налагает определенные ограничения на конструкции строк программы. Например, нельзя передать управление внутреннему оператору, минуя первый оператор строки.

При вводе программы, состоящей из большого количества строк, после заполнения всего экрана ее текст будет сдвигаться вверх. При этом верхняя строка исчезнет, однако она не пропадет совсем. Вычислительная машина сохранит весь текст программы в своей памяти. Пользователь с помощью системной команды LIST может вывести на экран введенный текст программы и проверить, нет ли ошибок. Эту команду можно задать и в виде

LIST m1—m2

где m1 — начальный и конечный номера строк.

По данной команде на экране появятся строки с номерами от m1 до m2. Например, команда LIST 10—50 выведет на экран дисплея строки с номерами от 10 до 50.

При вводе программы в ЭВМ пользователь может ошибиться. Ошибку во введенной строке можно легко исправить. Для этого надо заново набрать строку с тем же самым номером. При этом старая строка стирается, а вновь набранная строка запишется на ее месте в памяти машины. Более подробно средства редактирования, отладки и управления выполнением программы см. в гл. 6.

1.7. ПРОГРАММНЫЙ И НЕПОСРЕДСТВЕННЫЙ РЕЖИМЫ

БЕЙСИК-система может выполнять операторы в двух режимах: программном и непосредственном. В программном режиме каждая строка программы начинается с номера, указывающего на последовательность выполнения операторов. В этом режиме выполнение операторов начинается после подачи специальной инструкции. В непосредственном режиме операторы языка БЕЙСИК набираются в строке без номера. Выполнение оператора осуществляется сразу же после нажатия клавиши возврата каретки <ВК>. Результат выполнения хранится для последующего использования, но оператор теряется. Этот режим аналогичен режиму работы обычного калькулятора.

Непосредственный режим необходим при редактировании и отладке программ, записанных в программном режиме. Он может использоваться для проверки некоторых операторов, для печати текущих значений переменных в процессе выполнения программы. Например, строку программы 10PRINT5+6 БЕЙСИК-система поместит в память для более позднего исполнения. Но если набрать ее без номера, она выполнит строку и выдаст результат и сообщение READY:

```
PRINT 5+6
 11
READY
```

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Что такое алгоритм и схема алгоритма?
2. Что такое оператор и программа?
3. Из каких устройств состоит микроЭВМ?
4. Что такое БЕЙСИК-система?
5. Что такое числовая константа и какие их типы вы знаете?
6. Что такое строковая константа?
7. Что такое переменная и для чего служит имя переменной?
8. Какие стандартные функции имеются в языке БЕЙСИК?
9. Что такое арифметическое выражение и каковы правила их записи?
10. Как записывается программа на языке БЕЙСИК?
11. Что такое номер строки и для чего он нужен?
12. Каково назначение системных команд?

Глава 2

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ «БЕЙСИК»

2.1. ОПЕРАТОР ПРИСВАИВАНИЯ LET

Операторы языка БЕЙСИК можно разделить на группы, позволяющие: 1) вычислять значения и присваивать их переменным; 2) вводить информацию и выдавать результаты и сообщения; 3) управлять ходом вычислений. Рассмотрим более подробно каждую группу операторов.

В процессе вычислений на ЭВМ переменная величина может принимать различные числовые значения. В частности, ей может быть присвоено значение вычисленного арифметического выражения с помощью специальной операции присваивания, которая в языке БЕЙСИК обозначается знаком «=». Наименование переменной записывается слева, арифметическое выражение — справа от этого знака. Для выполнения этой операции имеется специальный оператор присваивания. Общая форма оператора

LET v = e

где LET — имя оператора (*пусть*); v — имя переменной; e — арифметическое выражение (в частном случае может быть числовая константа или переменная).

Оператор присваивания выполняется в два этапа: 1) вычисляется значение выражения e; 2) вычисленное значение e присваивается переменной v, т. е. засылается в ячейку памяти, отведенную для v.

Если e является числовой константой или переменной, то первый этап пропускается. Следовательно, к моменту выполнения оператора LET значения всех переменных, входящих в правую часть, т. е. в выражение e, должны быть определены.

Примеры операторов присваивания:

```
30 LET A=10
50 LET B=C2
70 LET Y1=B*SQR((Y*Z)-2)+3.5
```

Следует обратить внимание на существенную разницу между знаком присваивания «=» в языке БЕЙСИК и обычным знаком равенства. Например, в обычной математической записи выражение X = X + 1 является неверным. В языке БЕЙСИК запись

10 LET X = X + 1

справедлива и означает следующее:

1) берется значение переменной из ячейки памяти с именем X;

2) это число складывается с единицей и образуется новое значение, равное X+1;

3) результат X+1 помещается в ячейку памяти с именем X, стирая находившееся там прежнее значение.

Перед выполнением оператора LET в ячейке X находилось число, равное 5. После его выполнения в ячейке X будет находиться число, равное 6.

Во фрагменте программы, приведенном ниже,

```
200 LET A=5
210 LET B=2
220 LET A=(A+B)^2
```

после выполнения оператора с номером 220 переменной A будет присвоено значение 49.

При записи оператора пробелы можно использовать по своему усмотрению:

```
30LETT2=10
30 LET T2=10
30 L E T  T 2 =1 0
```

Использование пробелов должно улучшить наглядность программы.

В некоторых версиях языка БЕЙСИК, например в БЕЙСИКе ДВК и MSX, наряду с традиционной записью оператора присваивания разрешается опускать имя оператора LET. Например,

```
10 A = 25*SIN (X)
```

2.2. ОПЕРАТОРЫ ЗАДАНИЯ ЗНАЧЕНИЙ ПЕРЕМЕННЫМ

Задание значений переменным можно осуществить с помощью группы оператора DATA, READ, RESTORE.

Оператор DATA. В отличие от оператора LET задание значений переменных может быть основано на использовании блока данных. Присвоить значения переменным можно путем чтения чисел или строковых констант из этого блока. Он представляет собой упорядоченный числовой массив, формируемый в памяти ЭВМ перед началом выполнения программы. Числовая информация, необходимая для создания блока данных, задается в исходной программе с помощью оператора DATA. Общая форма оператора

```
DATA c1, c2, ..., cN
```

Здесь DATA — имя оператора (*данные*); c1, c2, ..., cN — константы (*числа*), разделяемые запятыми.

Например,

```
10 ДАТА —1, 0.86, 03E—5
```

По этому оператору перед началом счета будет сформирован блок данных, состоящий из трех чисел: —1 — первое, 0,86 — второе и $0{,}3 \cdot 10^{-5}$ — третье числа.

Операторы DATA принадлежат к группе невыполняемых операторов и могут быть записаны в любом месте программы.

При решении на ЭВМ задачи по программе, составленной на языке БЕЙСИК имеют место два отдельных этапа:

1) ввод операторов программы в вычислительную машину и преобразование вводимых операторов в форму, понятную машине. На этом этапе формируется блок данных, заданный оператором DATA;

2) выполнение операторов программы в порядке возрастания их номеров. Если при выполнении операторов встретится оператор DATA, то он никакого действия на ход вычислений не оказывает, так как формирование блока данных произошло на предыдущем этапе. В этом смысле оператор DATA и является невыполняемым.

В исходной программе может быть несколько операторов DATA, однако из них формируется только один блок данных. Например,

```
5   DATA 2,-1, 0
...
20 DATA 0.3, -71.5
...
100 DATA 10, 12
```

В этом случае блок данных формируется в порядке очередности операторов DATA. Сформированный по ним блок данных будет включать семь чисел в следующем порядке: 2; —1; 0; 0,3; —71,5; 10; 12.

Оператор DATA нельзя включать в строку с другими операторами; он должен быть единственным оператором на нумерованной строке.

Оператор DATA формирует данные, которые могут быть считаны с помощью оператора READ.

Оператор READ. Он является оператором выборки значений из блока данных. Общая форма оператора

READ v1, v2, ..., vN.

Здесь READ — имя оператора (*читать*); v1, v2, ..., vN — имена переменных.

С помощью этого оператора на этапе выполнения программы числовая информация извлекается из блока данных и присваивается переменным. Например, имеется фрагмент программы:

```
10 DATA 6,-10,1.2,100
20 READ X,Y,Z
...
150 READ A
```

Во время выполнения оператора 20 переменной X будет присвоено значение 6, переменной Y — значение —10, а переменной Z — значение 1,2. После этого в блоке данных остается неиспользованным только одно число — 100. Оно может быть выбрано впоследствии при выполнении другого оператора READ, например 150 READ A.

Блок данных допускает только последовательную выборку. Нельзя, например, прочитать из него сразу третье число. Попытка прочитать число из опустевшего блока данных, т. е. такого, из которого уже извлечены все числа, считается ошибкой. БЕЙСИК-система печатает знак «?» и выполнение программы прекращается.

Оператором READ удобно пользоваться для формирования начальных значений переменных. Это экономит общее количество операторов в программе. Для изменения начальных параметров потребуется лишь заменить небольшое количество операторов DATA.

В версиях языка, допускающих запись нескольких операторов в строке, оператор READ можно записать в любом месте.

Перед выполнением программы БЕЙСИК-система просматривает все операторы DATA в порядке их появления и создает блок данных. Каждый раз, когда в программе встречается оператор READ, БЕЙСИК-система последовательно берет значения из блока данных и присваивает их переменным оператора READ, которые также берутся последовательно. После выполнения этого оператора положение последнего считанного данного запоминается с помощью особого указателя, устанавливаемого на очередном элементе данных. Следующий оператор READ начинает выбирать данные с этого элемента.

Оператор RESTORE. Он служит для восстановления блока данных в исходное состояние в тех случаях, когда из блока данных уже выбраны значения. Общая форма оператора

RESTORE

Здесь RESTORE — имя оператора (*восстановить*).

После выполнения этого оператора чтение чисел операторами READ начинается с первого числа блока данных. Оператор RESTORE может быть выполнен в любой момент работы программы, т. е. и до того, как из блока данных будут считаны все числа. Во фрагменте программы

```
10 DATA 40, 0.3, 2.8, -0.9
20 READ M,A,B,C
...
100 RESTORE
120 READ A,A,B,C
...
```

при выполнении оператора 20 переменным будут присвоены значения: M = 40; A = 0,3; B = 2,8; C = —0,9. В процессе решения задачи эти переменные могут приобрести какие-то другие значения.

Пусть далее требуется вновь присвоить переменным A, B и C те же значения, не восстанавливая значения переменной M. Для этого используется оператор RESTORE, подготавливающий чтение чисел из блока данных, начиная с первого числа. Чтобы пропустить присваивание первого числа переменной M, переменная A в операторе 120 указана дважды. При этом первое присваивание значения переменной A является фиктивным, а следующее за ним второе присваивание — правильным.

Оператор RESTORE позволяет повторно использовать константы

оператора DATA, начиная с меньшего номера строки оператора DATA.

Оператор RESTORE может быть единственным оператором на строке или включаться в строку с несколькими операторами (в версиях языка, допускающих запись нескольких операторов в строке).

2.3. ОПЕРАТОРЫ ВВОДА — ВЫВОДА ИНФОРМАЦИИ

Операторы ввода — вывода информации предназначены для ввода информации в программу и выдачи результатов вычислений пользователю.

Оператор ввода данных INPUT. Он позволяет пользователю вводить данные с клавиатуры в процессе выполнения программы. При появлении оператора INPUT программа делает запрос на ввод данных с терминала.

Общая форма оператора

INPUT v1, v2, ..., vN

где INPUT — имя оператора (*ввести*), v1, v2, ..., vN — список переменных.

Например, 10 INPUT A, B, C,

Выполняя оператор INPUT, машина выводит знак вопроса на экран дисплея и ждет ввода значений для каждой переменной в списке этого оператора. Пользователь должен в данном случае набрать три числа, разделяя их запятыми, и нажать клавишу <ВК>, т. е. ввести их в ЭВМ. Например,

? 45.1, —2, 1.1 E—4 <ВК>

После этого переменные примут следующие значения: A = 45,1, B = —2; C = $1{,}1 \cdot 10^{-4}$. Далее выполнение программы будет продолжено с оператора, следующего за оператором INPUT. Таким образом, действия операторов INPUT и READ аналогичны, с той лишь разницей, что в операторе INPUT данные поступают с клавиатуры дисплея, а не из блока данных.

Набирая числа для ввода, надо следить за тем, чтобы их количество и тип точно совпадали с количеством и типом переменных, указанных в списке оператора INPUT. Если набрать их слишком много, то лишние числа проигнорируются и будет выдано предупреждение об ошибке.

Если ввести недостаточное количество чисел, то машина снова напечатает знак вопроса и будет ждать ввода недостающих значений. Например, возможны такие варианты ввода значений при выполнении оператора 10 INPUT A, B, C:

```
        ? 45.1 — 2 <ВК>
        ? 1.1 E—4 <ВК>
        Продолжение программы
либо    ? 45.1      <ВК>
        ? —2        <ВК>
        ? 1.1 E—4   <ВК>
        Продолжение программы
```

При вводе значений

? 10%, 25%, 37% <ВК>

будет выдано сообщение об ошибке из-за несоответствия типов.

Приведем еще один пример выполнения оператора для другой БЕЙСИК-системы («Электроника-60»):

10 INPUT A, B, C

БЕЙСИК-система делает паузу во время выполнения, печатает знак вопроса и ждет, когда пользователь введет три числовых значения. Для каждого оператора INPUT печатается только один знак вопроса.

Если введено недостаточное количество данных, БЕЙСИК-система, например, печатает

ОШИБКА 121 В СТРОКЕ 10

Если введено слишком много данных, БЕЙСИК-система печатает

ОШИБКА 122 В СТРОКЕ 10

В обоих случаях БЕЙСИК-система печатает второй знак вопроса и ждет выполнения повторного ввода. Строка ввода должна быть набрана вновь.

Оператор INPUT может записываться в любом месте программы. Он может быть единственным оператором в строке или включаться в нее с несколькими операторами (в версиях языка, допускающих запись нескольких операторов в строке). Так как этот оператор не может выполняться сразу, а ждет исходных данных с терминала, использование его в непосредственном режиме в некоторых версиях языка БЕЙСИК запрещено.

Оператор вывода PRINT *(печатать)*. Он осуществляет вывод на терминал результатов вычислений и пояснительных текстов. Общая форма оператора

PRINT<*выводной список*>

Элементы выводного списка разделяются либо запятой, либо точкой с запятой. Ими могут быть: константы, имена переменных, выражения, тексты, функция TAB (X). Например,

```
40 PRINT A,B,125;SIN(X).
50 PRINT "ЗНАЧЕНИЕ A=";A
60 PRINT TAB(25),1
```

Значения, выводимые на терминал, зависят от элементов выводного списка:

если в качестве элемента указана константа, то при выполнении оператора PRINT она будет напечатана на терминале;

если в качестве элемента указана переменная, то на терминале будет напечатано ее значение;

если в качестве элемента указано арифметическое выражение, то на терминале будет напечатан его результат.

Текст в списке оператора PRINT используется для печати заголовков и пояснений. Такой текст заключается в апострофы или кавычки

и может состоять из произвольной последовательности символов входного языка (включая и русский алфавит).

При выполнении оператора PRINT этот текст будет выдан на терминал без апострофов (навычек).

Использование функции TAB(X) в качестве элемента выводного списка будет рассмотрено позже.

При выводе из ЭВМ данные печатаются либо на устройстве печати либо на дисплее. Поэтому длина каждой выводимой строки ограничена конструкцией устройства.

Для удобства и наглядности расположения выводимых данных каждая выводимая строка условно делится на несколько зон равной длины. В диалоговом вычислительном комплексе ДВК-2 число таких зон равно 5, максимальная длина строки — 72 позиции, длина каждой зоны — 14 позиций. Нумерация позиций начинается с нуля.

При печати чисел (констант и значений переменных или выражений) обычная форма записи — по 5 чисел в строке, т. е. по одному числу в зоне. Указанием о такой форме печати является использование запятой в качестве разделителя элементов выводного списка. Например, оператор 200 PRINT A, B, SQR (A*B) приведет к выводу на печать трех чисел со следующим расположением в зонах: в первой зоне будет напечатано первое число — значение переменной A; во второй зоне — второе число — значение переменной B; а в третьей зоне — значение выражения $\sqrt{AB}$.

Форма печати того или иного числа зависит от его величины и типа, а также от типа ЭВМ, так как различные ЭВМ оперируют с различными форматами данных и соответственно имеют различные диапазоны представления чисел.

Для микроЭВМ ДВК-2 форма печати чисел следующая:

а) результат вычисления в диапазоне 0,01—999999 печатается в виде десятичного числа (целого или содержащего десятичную точку) без указания порядка;

б) значения, по абсолютной величине меньшие 0,01 и большие 999999, печатаются в форме с десятичным порядком.

В обоих случаях печатается максимум шесть значащих цифр. Например, при выполнении оператора

```
20 PRINT 0.00789; 0.25,888889,1234566
```

на экране дисплея напечатается строка

```
7.89000E-03   .25            888889          1.23457E+06

READY
```

Если в выводном списке оператора указано более пяти числовых величин, разделенных запятыми, то первые пять из них будут напечатаны в одной строке, последующие пять — во второй и т. д.

При зонной печати числа прижимаются к левому краю соответствующей зоны. При печати чисел первая позиция в зоне отводится для знака числа. Если знак «+», в ней печатается пробел. Например,

```
5 PRINT 1,2,3,4,5,-6,7,8,9,10,11
10END
RUN

 1            2            3            4            5
-6            7            8            9            10
 11
```

Таким образом, использование запятой в качестве разделителя всегда вызывает печать очередного числа в следующей зоне либо с новой строки, если на предыдущей строке заполнены все пять зон.

Более плотную печать можно получить, если употреблять точку с запятой в качестве разделителя в выводном списке оператора PRINT. Точка с запятой в качестве разделителя вызывает печать очередного числа через один пробел после предыдущего числа. Перед положительными числами вместо знака «+» печатается еще один пробел:

```
5 PRINT 1; 2; —3; —4; 5; 6; 7; 8; 9; 10
10 END
RUN
1 2 —3 —4 5 6 7 8 9 10
```

Если в текущей строке для печати очередного элемента списка не хватает места, то он начинает печататься с начала следующей строки.

В программе может быть несколько операторов печати PRINT. В этом случае печать с начала новой строки при выполнении очередного оператора печати зависит от разделителя (запятая, точка с запятой или отсутствие знака), которым завершен выводной список предыдущего оператора печати:

1) если в конце списка предыдущего оператора PRINT знак разделителя отсутствует, то следующий выполняемый за ним оператор PRINT начинает печатать значения с новой строки. Например,

```
5 LET A=-35\B=26\C=-46\D=256
...
40 PRINT A,B;
...
80 PRINT C;D

-35            26 -46   256
```

2) если список предыдущего оператора печати закончился запятой, то следующий выполняемый за ним оператор PRINT начинает печатать значение с новой зоны текущей строки. Например,

```
5 LET A=-35\B=26\C=-46\D=256
...
40 PRINT A,B,
...
80 PRINT C;D

-35            25             -46   256
```

3) если список предыдущего оператора печати закончился точкой с запятой, то следующий выполняемый оператор PRINT продолжает печатать информацию в текущей зоне с пропуском одного пробела за последним напечатанным элементом. Например,

```
5 LET A = -35\B = 26\C = -46\D = 256
  . . .
40 PRINT A, B;
  . . .
80 PRINT C; D
-35          26 -46  256
```

Таким образом, действие операторов

```
105 PRINT A,B,
110 PRINT SQR(A*B)
```

эквивалентно оператору

```
120 PRINT A,B,SQR(A*B)
```

Такой прием (разбиение выводного списка) можно применять, когда выводной список не помещается на одной строке.

Отсутствие запятой или точки с запятой в конце выводного списка всегда вызывает после печати переход к началу новой строки. Этим правилом можно пользоваться для организации пропуска на печатном документе нужного числа строк. С этой целью следует повторить несколько раз оператор PRINT без выводного списка. Для пропуска зоны достаточно поставить дополнительную запятую. Например,

```
READY
10 PRINT 1,2,,3;4%
20 PRINT
30 PRINT
40 PRINT "ПЕРЕД ЭТИМ ТЕКСТОМ ПРОПУЩЕНО 2 СТРОКИ"
50 PRINT 1,
60 PRINT 2
70 END
RUN

 1             2                                3  4


ПЕРЕД ЭТИМ ТЕКСТОМ ПРОПУЩЕНО 2 СТРОКИ
 1             2

READY
```

При печати целых чисел знак % не печатается.

Функция TAB(X) в операторе PRINT. Дополнительные возможности при печати обеспечивает функция TAB (X), аргументом которой является выражение или число. Значение функции равно целой части аргумента. Оно указывает позицию, в которую будет выводиться значение очередного элемента, записанного за функцией TAB (X). Позиции строки нумеруются от 0 до 71. В приведенном ниже примере выводимое значение A = —35 начинает печататься с пятой позиции:

```
100 LET A=-35
110 PRINT TAB(5);"A=";A
RUN

     A=-35

READY
```

Если значение функции TAB (X) <0, то указанный за ней элемент будет печататься с нулевой позиции текущей строки; если значение функции TAB (X) >71, то указанный за ней элемент будет печататься в позиции, вычисляемой следующим образом: X делится на 72 и выделяется частное и остаток. Целое частное означает число пропущенных строк. Остаток указывает номер позиции в строке.

Если строка печати частично заполнена и функция TAB (X) указывает на уже занятую позицию, то эта функция игнорируется. Например,

```
100 LET A=-35 \ B=26
110 PRINT TAB(5);"A=";A;TAB(8);B
RUN

     A=-35   26

READY
```

Здесь функция TAB (8) не учитывается.

Разделители, стоящие после функции TAB (X), не влияют на ее действие. Так, запятая не вызывает печати очередного элемента списка в следующей зоне. Например, оператор

20PRINT TAB (7), A + B

вызывает печать значения A + B, начиная с позиции 7, между тем как оператор 20 PRINT Z, A + B вызовет печать этого значения, начиная с позиции 14.

2.4. ОПЕРАТОР REM

Оператор REM. Он применяется для ввода комментариев и примечаний в программу и, в частности, для указания наименования программы (REM — начальные буквы слова remark — *комментарий*). Комментарии могут быть составлены из любых допустимых символов языка БЕЙСИК, включая буквы русского алфавита. Общая форма оператора

REM ⟨*текст*⟩

Ниже приведен пример использования этого оператора:

```
10 REM ПРОГРАММА ВЫЧИСЛЕНИЯ СИНУСА
20 REM ВВОД АРГУМЕНТА С ДИСПЛЕЯ
30 INPUT X
```

```
40 LET Y=SIN(X)
50 REM ВЫВОД РЕЗУЛЬТАТА
60 PRINT "Y=";Y
70 END
RUN

? 3
Y= .14112

READY
```

Операторы REM целесообразно использовать для облегчения чтения и понимания текста программы. Они относятся к группе невыполняемых и поэтому не влияют на ее выполнение. При написании больших программ необходимо учитывать, что операторы REM с текстом комментария занимают место в памяти машины. При использовании оператора REM на одной строке с другими операторами он должен записываться последним, так как БЕЙСИК-система игнорирует все, что будет в строке после REM.

2.5. ОПЕРАТОРЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

Обычно операторы программы выполняют в порядке возрастания их номеров строк. Это естественный порядок выполнения программы. Однако большинство задач редко укладывается в такую линейную схему. Часто в задачах встречаются условия, от выполнения которых зависит последующий ход вычислений. Чтобы изменить естественный порядок выполнения операторов программы, используют операторы передачи управления. К ним относятся операторы безусловного и условного переходов, а также оператор-переключатель.

Оператор безусловного перехода GO TO. Оператор безусловного перехода GO TO (*идти к*) используется для указания того места программы, т. е. того оператора, с которого следует продолжать вычисления на данном этапе. Как правило, этот оператор применяется для обхода какого-то участка программы. Общая форма оператора

GO TO m1

где GO TO — имя оператора (*идти к*); m1 — номер строки с оператором, который будет выполняться за оператором GO TO.

Пробел в операторе используется для удобства чтения. Например, оператор 10 GO TO 120 можно записать 10GOTO120. После оператора с номером 10 будет выполняться оператор с номером 120.

При записи в строке с несколькими операторами оператор GO TO должен быть последним в строке. Если это правило не соблюдено, то следующие за ним операторы никогда не будут выполнены, так как на них нельзя передать управление.

Оператор условного перехода IF. Часто при составлении алгоритма решения задачи приходится проводить анализ выполнения тех или иных

условий. Как правило, от выполнения таких условий зависит дальнейший ход решения задачи. Для изменения естественного порядка выполнения программы при выполнении некоторого условия используется оператор условного перехода IF. Общая форма оператора

IF e1 ⊗ e2 THEN m1

где IF — имя оператора (*если*); e1 ⊗ e2 — проверяемое условие; e1, e2 — арифметические выражения; ⊗ — условное обозначение знака операции отношения (>, <, >=, <=, =, <>); в частном случае слева и справа от знака операции отношения может стоять число или переменная; THEN (*тогда*) — служебное слово (вместо него допустимо писать GO TO); m1 — номер строки с оператором, который будет выполняться за оператором IF, если условие e1 ⊗ e2 выполнено (*истинно*).

Если условие e1 ⊗ e2 нарушено (*ложно*), сохраняется естественный порядок выполнения операторов, т. е. будет выполняться оператор, следующий за оператором IF. Пример записи оператора IF:

30 IF A > B THEN 120

Кроме стандартной формы оператор IF может иметь вид

IF e1 ⊗ e2 THEN <*оператор*>

где <*оператор*> — любой оператор языка БЕЙСИК, в том числе и условный.

Если условие e1 ⊗ e2 истинно, то выполняется оператор, следующий за словом THEN. Если условие ложно, то управление передается строке, следующей за строкой с оператором IF. Например, если во фрагменте программы

```
20 IF A+B<=10 THEN PRINT Z
30 LET C=2
...
```

A + B > 10, то оператор PRINT Z не выполняется, а выполняется оператор с номером 30.

В некоторых версиях языка БЕЙСИК в операторе IF вместо условия e1 ⊗ e2 может быть записано одно арифметическое выражение e1. В этом случае при выполнении оператора вычисляется значение арифметического выражения e1, и если результат его равен нулю, то это понимается как нарушение условия, т. е. «*ложь*». Если результат вычисления e1 не равен нулю, то это понимается как выполнение условия, т. е. «*истина*». Далее оператор выполняется, как обычно. Например,

```
70 LET A=3
80 LET B=4
90 IF A+B-5 THEN 300
```

Здесь результат выражения 3 + 4 — 5 = 2, т. е. не нуль («*истина*») и, следовательно, после оператора 90 будет выполняться оператор с меткой 300.

В версиях языка, допускающих запись нескольких операторов в строке, оператор IF может записываться в любом месте на строке, за исключением случая, когда за THEN следует оператор GOSUB или

GO TO. В этом случае оператор IF должен быть последним оператором на строке.

Программирование алгоритмов разветвляющейся структуры. Такие алгоритмы программируют с помощью операторов передачи управления.

Пример 2.1. Составим программу нахождения и печати наибольшего из двух чисел X и У.

```
5 REM НАХОЖДЕНИЕ БОЛЬШЕГО ЧИСЛА
10 INPUT X,Y
20 IF Y>X THEN 50
30 PRINT "X=";X
40 GO TO 60
50 PRINT "Y=";Y
60 END
RUN

? 12.5,6.3
X= 12.5

READY
```

После ввода значений переменных X и Y оператор IF проверяет условие $Y > X$. Если это условие удовлетворяется, то следующим выполняемым оператором будет оператор с номером 50, печатающий значение Y, являющееся большим. Затем выполняется следующий по порядку оператор END (*конец*) с номером 60, и работа программы заканчивается.

Если условие $Y > X$ не удовлетворяется, то после оператора IF с номером 20 будет выполняться следующий за ним оператор 30, печатающий значение переменной X, которое в этом случае является не меньшим Y. Оператор IF при составлении программы записывают вначале не полностью, т. е. без указания номера m1. Этот номер дописывают позже, когда нужный участок программы будет составлен.

Следует обратить внимание на обязательность использования оператора 40 GO TO 60, который разделяет два оператора печати. Без него после оператора с номером 30 выполнялся бы оператор с номером 50, т. е. были бы отпечатаны оба числа, несмотря на требование печати только наибольшего числа.

Пример 2.2. Требуется вычислить

$$y = \begin{cases} x + z, & \text{если } x < 5, \\ 2x - 3z, & \text{если } 5 \leqslant x \leqslant 10, \\ x + z^2, & \text{если } x > 10. \end{cases}$$

Здесь вычислительный процесс разветвляется на три направления. Составим схему алгоритма программы (рис. 2.1). На схеме принято обозначать: овалом — начало и конец программы, прямоугольником — действия, которые следует выполнять на данном шаге вычислений, ромбом — место разветвления вычислений. В ромбе записывается условие, определяющее порядок выполнения дальнейших действий. Из ромба выходят две стрелки. Над одной из них записывается слово ДА, над другой — НЕТ. Если условие, записанное в ромбе, выполнено, выход

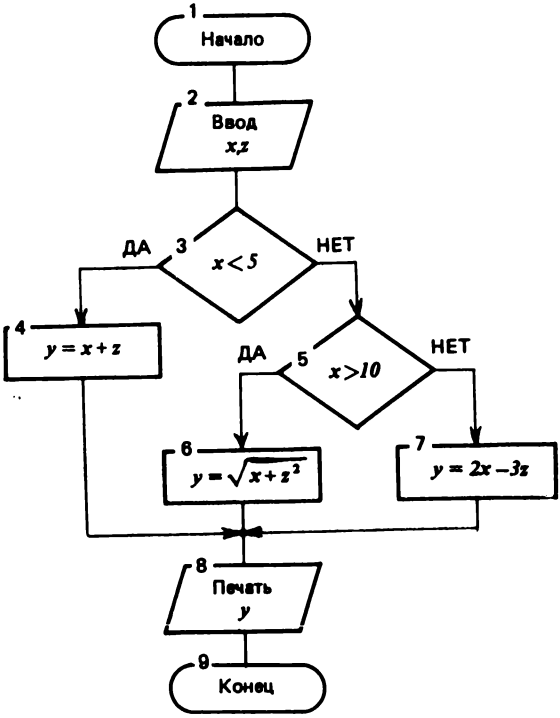


Рис. 2.1.
Схема алгоритма к примеру разветвляющегося вычислительного процесса

производится по стрелке ДА, если условие не выполнено — по стрелке НЕТ. Ввод и вывод информации обозначают параллелограммом. Направление хода вычислительного процесса указывается линиями (со стрелками и без них). Программа имеет вид:

```
40 REM ПРИМЕР ВЫЧИСЛЕНИЯ С РАЗВЕТВЛЕНИЕМ
50 INPUT X,Z
60 IF X<5 THEN 120
70 IF X>10 THEN 100
80 LET Y=2*X-3*Z
90 GO TO 130
100 LET Y=SQR(X+Z^2)
110 GO TO 130
120 LET Y=X+Z
130 PRINT "Y=";Y
140 END
RUN
? 7,4.5
Y= .5

READY
```

Здесь операторы 90 и 110 разделяют вычисление по формулам и являются обязательными. Без них после оператора с номером 80 выполнялся бы оператор с номером 100, а затем с номером 120 и значение Y, вычисленное по одной формуле, замещалось бы новым значением,

вычисленным по второй формуле, а затем аналогично значением по третьей формуле. Такой результат, очевидно, не соответствует условию задачи и является грубой ошибкой.

Благодаря разделению ветвей вычислений с помощью операторов 90 и 110, которые передают управление оператору PRINT с меткой 130, на печать будет выведено вычисленное по одной из трех заданных формул значение Y в соответствии с введенным значением X.

Программирование циклических вычислительных процессов с применением операторов передачи управления. Общей чертой программ для ЭВМ (не только программ на языке БЕЙСИК) является возможность неоднократного выполнения одной и той же последовательности операторов внутри программы. Такая неоднократно выполняемая последовательность операторов называется *циклом*. Можно выделить два основных типа цикличности вычислительных процессов:

многократное повторение вычислений по одной и той же формуле (формулам) при различных исходных данных. Например, вычисление $y = x^2$, если $x = 1, 2, 3, 4, \ldots$;

вычисление некоторых переменных в данном цикле через значения этих же переменных, полученных в предыдущем цикле. Например, возведение числа a в целочисленную степень n. Этот процесс может быть представлен следующей вычислительной схемой:

$$a^1 = a^0 \quad a$$
$$a^2 = a^1 \quad a$$
$$a^3 = a^2 \quad a$$
$$\ldots$$
$$a^n = a^{n-1} \quad a$$

где $a^0 = 1$.

Если предыдущее значение степени обозначить символом P_{i-1}, а последующее — символом P_i, то вычислительная схема может быть представлена формулой

$$P_i = P_{i-1}a,$$

где i должно меняться от 1 до n.

Данная формула, в которой последующее значение величины вычисляется через предыдущее, называется *рекуррентной*. Чтобы выполнить вычисления по рекуррентной формуле, необходимо рекуррентной (вычисляемой) величине задать начальное значение в данном случае $P_0 = 1$.

Приведение алгоритма задачи к рекуррентной вычислительной схеме — прием, широко распространенный в технике программирования.

Многократное примение оператора языка БЕЙСИК

```
40 LET P=P*A
```

при начальном значении P = 1 и заданном значении A позволяет постепенно накапливать возрастающую степень числа A в ячейке P. Операция присваивания, в которой справа и слева от знака присваивания находится одна и та же переменная, и реализует рекуррентную формулу. Она указывает, что из некоторой ячейки P надо взять число (преды-

дущее), умножить на А (получив последующее число) и записать результат снова в ячейку Р. При очередном выполнении оператора 40 значение в ячейке Р снова окажется предыдущим и т. д. Когда же следует остановиться?

Важной особенностью циклических программ является обеспечение своевременного выхода из цикла. Цикл характеризуется параметром и эталоном. *Параметр цикла* — это некоторая переменная (обычно целого типа), которая изменяется при выполнении каждого цикла. *Эталон цикла* — это некоторая постоянная величина, характеризующая задаваемый цикл (как правило, это конечное значение параметра цикла). При каждом выполнении цикла параметр цикла сравнивается с эталоном цикла и, как только они окажутся в заданном соотношении, производится выход из цикла.

По способу выхода из цикла циклические алгоритмы могут быть разделены на арифметические и итерационные.

Цикл называется *арифметическим*, если количество его повторений заранее известно или может быть легко вычислено. Такой цикл называют также циклом на достижение заданного значения.

Рис. 2.2. Обобщенная схема циклического алгоритма

Цикл называется *итерационным,* если число его повторений заранее не известно. Выход из таких циклов осуществляется, как правило, по достижении заданной точности вычислений. Итерационные циклы соответствуют различным сходящимся математическим процессам.

Циклические алгоритмы позволяют существенно сократить объем программы, что уменьшает затраты времени на ее составление и эксплуатацию машины. Обобщенная схема циклического алгоритма показана на рис. 2.2. Рассмотрим некоторые примеры.

Пример 2.3. Составить программу вычисления и печати квадратов всех чисел натурального ряда от 1 до 10. Программа будет иметь вид:

```
90 REM ВЫЧИСЛЕНИЕ КВАДРАТОВ ЧИСЕЛ
100 LET N=1                          Подготовка цикла
200 LET S=N^2
300 PRINT "N=";N,"S=";S              Рабочая часть цикла
400 LET N=N+1                        Подготовка к очередному исполнению цикла

500 IF N<=10 THEN 200                Проверка окончания цикла
600 END
RUN
```

```
N=  1          S=  1
N=  2          S=  4
N=  3          S=  9
N=  4          S=  16
N=  5          S=  25
N=  6          S=  36
N=  7          S=  49
N=  8          S=  64
N=  9          S=  81
N=  10         S=  100

READY
```

Из данной программы видно, что в ней содержатся оба вида цикличности:

оператор 200 — это многократное вычисление по формуле с различными исходными данными;

оператор 400 — вычисление по рекуррентной формуле, т. е. последующего значения через предыдущее.

Поясним работу программы. Сначала переменной N присваивается значение 1. Затем N возводится в квадрат и результат присваивается переменной S. По оператору 300 печатаются значения N и S. Далее к значению N прибавляется 1 и образуется новое значение N=2. Оператор IF позволяет вычислительной машине принимать решение. Когда исполнение программы доходит до оператора IF, ЭВМ проверяет условие, содержащееся в нем, т. е. значение переменной N сравнивается с числом 10. Если условие выполнено, т. е. $N \leqslant 10$, тогда ЭВМ переходит к строке 200 и снова выполняются те же самые операторы. Операторы в строках 200, 300, 400 и 500 составляют циклическую часть программы или тело цикла.

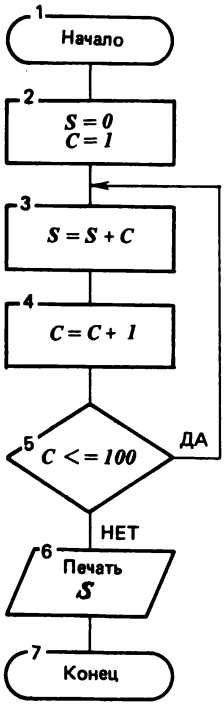


Рис. 2.3. Схема алгоритма к примеру циклического вычислительного процесса.

Только при N=11, т. е. когда нарушится условие, произойдет передача управления к строке 600 и вычисления прекратятся.

В результате работы программы будет напечатана таблица из двух колонок, содержащих по 10 чисел (значение числа и его квадрата).

Пример 2.4. Вычислить сумму чисел от 1 до 100, т. е. $S=1+2+3+...+100$.

Для решения задачи используем рекуррентные соотношения:

$$S = S + C;\ C = C + 1,$$

где S — текущее значение суммы чисел; C — текущее значение числа (начальные значения S=0 и C=1).

Схема алгоритма решения примера приведена на рис. 2.3. Программа, составленная в соответствии с данным алгоритмом, имеет вид:

```
90 REM ВЫЧИСЛЕНИЕ СУММЫ ЧИСЕЛ
100 LET S=0
200 LET C=1
300 LET S=S+C
400 LET C=C+1
500 IF C<=100 THEN 300
600 PRINT "СУММА ЧИСЕЛ"
700 PRINT "S=";S
800 END
RUN

СУММА ЧИСЕЛ
S= 5050

READY
```

Здесь при каждом выполнении цикла в переменной S будет накапливаться сумма. В результате работы программы после 100-кратного выполнения цикла будет напечатано одно значение, равное сумме первых ста чисел натурального ряда.

Следует отметить, что при составлении программ с разветвлениями оператор IF передает управление вперед, т. е. к операторам с большими, чем у IF, номерами, а при программировании циклов он передает управление назад к операторам с меньшими, чем у IF, номерами.

Оператор-переключатель ON. Оператор условного перехода IF организует передачу управления по двум направлениям, т. е. позволяет реализовать только одно ответвление от «ствола» программы. Для организации нескольких ответвлений от одной точки служит оператор-переключатель ON. Он позволяет осуществлять переход к одной из нескольких указанных строк, в зависимости от значения арифметического выражения. Общая форма оператора

ON e GO TO m1, m2, ..., mK

Здесь ON — имя оператора (*по*); e — переключающее арифметическое выражение, которое в частном случае может быть простой переменной; GO TO — служебное слово (*идти к*); вместо слова GO TO допустимо писать THEN; mI — номер строки первого оператора I-й ветви (I = = 1, 2, ..., K).

Переключающее арифметическое выражение обычно стараются подобрать таким образом, чтобы оно принимало значения 1, 2, ..., K.

При выполнении оператора ON вычисляется значение переключающего выражения e и выделяется его целая часть I, используемая в качестве указателя на один из перечисленных номеров строк в списке. Если I = 1, то управление передается оператору с номером строки m1, т. е. первой ветви, если I = 2, то — оператору с номером строки m2 и т. д. Если вычисленное значение I меньше 1 или больше количества номеров строк в списке, то БЕЙСИК-система печатает сообщение об ошибке и выполнение программы прекращается. Например, оператор

40 ON P GO TO 210, 20, 60, 240

действует следующим образом: если P = 1, то переходят к оператору с меткой 210; если P = 2 — к оператору с меткой 20; если P = 3 — к оператору с меткой 60; если P = 4 — к оператору с меткой 240.

Если в этом примере для оператора ON нарушить условие $1 \leqslant P \leqslant 4$, то выполнение программы прекратится из-за ошибки.

Выигрыш от использования оператора-переключателя ON по сравнению с использованием нескольких операторов IF прямо пропорционален числу путей, на которые разветвляется программа в той или иной точке. Так, действие приведенного в примере оператора ON эквивалентно действию четырех операторов IF:

```
40 IF P=1 THEN 210
41 IF P=2 THEN 20
42 IF P=3 THEN 60
43 IF P=4 THEN 240
```

Операторы STOP и END. Эти операторы используются для останова и завершения выполнения программы. Оператор END (*конец текста*) является последним в программе. Оператор STOP (*останов*) вызывает останов программы и вывод сообщения:

```
STOP AT LINE m      (останов на строке m)
READY               (готов)
```

После останова программы по оператору STOP БЕЙСИК-система переходит в непосредственный режим. Останов программы позволяет пользователю вывести на терминал значение переменных, изменить их значения, т. е. отлаживать программы с помощью оператора STOP. Количество используемых операторов STOP неограниченно. Они могут размещаться в любом месте программы. Чтобы во время выполнения программа останавливалась, оператор STOP помещают в критические точки.

Если в данной точке выполнение программы оказывается правильным, то она может быть продолжена оператором GO TO в непосредственном режиме, указав номер строки, с которой необходимо продолжить программу.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Как выполняется оператор присваивания?
2. Что такое блок данных?
3. Как выполняется оператор READ?
4. С помощью какого оператора восстанавливается блок данных?
5. Какой оператор предназначен для ввода данных с клавиатуры в процессе выполнения программы?
6. Для чего служит оператор PRINT и каковы правила его использования?
7. Для чего нужны операторы передачи управления?
8. Каковы основные типы цикличности вычислительных процессов?
9. Что такое рекуррентная формула?
10. Какие способы выхода из цикла вы знаете?
11. Для чего предназначен оператор ON?

Глава 3

ПРОГРАММИРОВАНИЕ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ МАССИВОВ И ОПЕРАТОРОВ ЦИКЛА

3.1. МАССИВЫ. ДЕЙСТВИЯ С МАССИВАМИ

Понятие массива. Переменные с индексами. Однотипные величины на языке БЕЙСИК можно объединить в массив. *Массив* представляет собой упорядоченный набор однородных элементов, имеющий одно общее имя. Имя массива строится по тем же правилам, что и имя простой переменной. Каждому элементу массива присваивается имя массива, за которым следует индекс. Поэтому элементы массива часто называют *переменными с индексами.*

По способу организации в языке БЕЙСИК различают одномерные, двумерные и многомерные массивы.

Элементы одномерных массивов снабжаются одним индексом, определяющим порядковый номер элемента в массиве, например:

A(1) — первый элемент массива A;

B1(5) — пятый элемент массива B1;

C(K) — K-й элемент массива C. Индексы помещаются в скобки после имени массива.

Элементы двумерных массивов снабжаются двумя индексами, первый из которых определяет номер строки, а второй — номер столбца, на пересечении которых расположен соответствующий элемент массива, например: A(0, 0) — элемент, расположенный в нулевой строке и нулевом столбце двумерного массива A; B3(2, 3) — элемент, расположенный во второй строке и третьем столбце двумерного массива B3.

В двумерных массивах индексы разделяются запятыми. В языке БЕЙСИК обычно индексы нумеруются, начиная с нуля, хотя имеются версии языка, в которых нумерации индексов начинается с единицы.

Вместо термина *«индекс»* иногда используется термин *«индексное выражение»*. Это объясняется тем, что числовое значение индекса может задаваться формулой. Никаких ограничений на сложность этих формул язык БЕЙСИК не накладывает. Таким образом, в качестве индекса может служить: константа (число), переменная, арифметическое выражение. Например,

$$A(10),\ B(3,\ J),\ C(SIN(X)+Z*2)$$

Если в качестве индексного выражения употребляется формула, то после вычисления ее значения производится округление до ближайшего целого. Значения индексов могут находиться в диапазоне от 0

до 32 767 (для ДВК). В случае выхода индекса за указанный диапазон печатается сообщение об ошибке.

Оператор DIM. Если в программе используются массивы, то они должны быть предварительно описаны. Это значит, что машине должна быть дана информация о структуре и размере массива. Для описания массивов используется оператор DIM (сокращение слова dimension — *размерность*). Общая форма оператора:

DIM A(*гм* A), B(*гм* B), ..., C(*гм* C)

где DIM — имя оператора; A, B, C — имена массивов; *гм* A, *гм* B, *гм* C — границы массивов A, B, C.

Границы массивов представляют собой одно или несколько целых чисел без знака. Количество этих чисел определяет размерность массива, а их величины определяют максимальное значение каждого индекса элементов соответствующего массива. Например, оператор

10 DIM A(4), B(1, 2)

означает, что массив A — одномерный, состоящий из элементов A(0), A(1), A(2), A(3), A(4), а массив B — двумерный, состоящий из элементов

B(0, 0) B(0, 1) B(0, 2)
B(1, 0) B(1, 1) B(1, 2)

и имеющий две строки и три столбца. По заданным границам массивов БЕЙСИК-система определяет и выделяет необходимое количество ячеек памяти для хранения этих массивов. Так, по оператору

10 DIM A(4), B (1, 2)

будет выделено 5 ячеек для хранения значений элементов массива A и 6 ячеек — для хранения значений массива B (с учетом нулевых значений индексов).

Оператор DIM является невыполняемым оператором. Его можно поместить в любом месте программы и в любой части многооператорной строки. Описание массивов должно предшествовать использованию элементов этих массивов в других операторах. Операторы DIM рекомендуется помещать в начале программы.

Язык БЕЙСИК допускает употребление элементов массивов и без предварительного их описания (неявное описание). Границы таких массивов принимаются равными 10, т. е. под неописанный одномерный массив выделяется 11 ячеек, а под неописанный двумерный массив — 121 (11×11) ячейка оперативной памяти. Если по условиям задачи количество элементов массивов больше 11 или 121, то отсутствие описания приводит к ошибкам во время счета и выдается соответствующее сообщение. На первых стадиях знакомства с программированием рекомендуется все используемые массивы описывать в операторе DIM.

В программе не следует использовать переменную и массив с одним и тем же именем. Например, A — переменная, A(5) — элемент массива. Недопустимо использовать одно и то же имя для одномерного и двумерного массивов.

Максимальное значение индекса, указываемого в операторе DIM,

не должно превышать 32 767 (для ДВК) и 255 (для ЭВМ «Электроника-60»).

Размещение массивов в памяти ЭВМ. Как было показано ранее, память ЭВМ можно представить в виде последовательного (линейного) расположения ячеек памяти. Одномерный массив также представляет собой линейную последовательность элементов. Поэтому он в памяти ЭВМ располагается в последовательных ячейках памяти в порядке увеличения индекса (рис. 3.1).

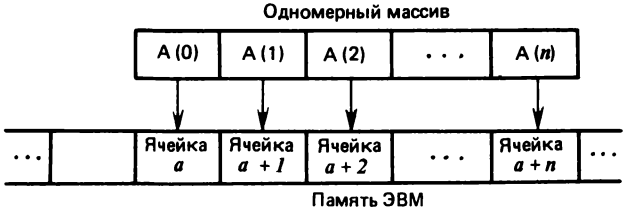


Рис 3.1. Размещение элементов одномерного массива в памяти ЭВМ:
A(*i*) — значение *i*-го элемента одномерного массива; *a* — адрес начальной ячейки памяти, отведенной для массива A

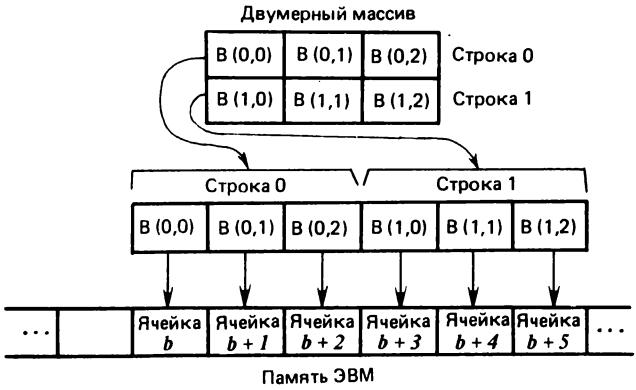


Рис. 3.2. Размещение элементов двумерного массива в памяти ЭВМ:
B(*i*, *j*) — значение элемента с индексом (*i*, *j*) двумерного массива B; *b* — адрес начальной ячейки памяти, отведенной для массива B

Двумерный массив (матрица) имеет два измерения (строки и столбцы). Однако он должен разместиться в линейной последовательности ячеек памяти. Перед размещением такой массив разворачивается в линейную форму, а затем записывается в память ЭВМ (рис. 3.2). В языке БЕЙСИК двумерный массив размещается в памяти по строкам: сначала первая строка, затем вторая и т. д. В пределах строки элементы размещаются в порядке увеличения второго индекса.

Формирование массивов. Массивы могут быть введены в ЭВМ как исходные данные с помощью операторов READ и INPUT и формироваться в программе как промежуточные или выходные результаты. В том и другом случаях все использованные в программе массивы следует описывать в операторах DIM. Рассмотрим формирование массива на следующем примере.

Пример 3.1. Вычислить значение первых десяти элементов геометрической прогрессии, если известен ее первый элемент $a_1 = 2$ и знаменатель прогрессии $q = 3$.

Известно, что каждый последующий элемент геометрической прогрессии образуется путем умножения предыдущего элемента на знаменатель прогрессии. Это удобно делать в цикле. Чтобы запомнить значения элементов, их следует объявить как одномерный массив. Программа имеет вид:

```
10 REM ВЫЧИСЛЕНИЕ ЭЛЕМЕНТОВ ПРОГРЕССИИ
20 DIM A(10)
30 LET A(1)=2
40 PRINT A(1)
50 LET Q=3
60 LET I=2
70 LET A(I)=A(I-1)*Q
80 PRINT A(I)
90 LET I=I+1
100 IF I<=10 THEN 70
110 END
RUN

 2
 6
 18
 54
 162
 486
 1458
 4374
 13122
 39366

READY
```

Здесь оператор 20 резервирует 10 ячеек памяти* для помещения туда последующих результатов вычисления элементов прогрессии.

Оператор 30 присваивает значение первому элементу прогрессии, а оператор 40 выводит это значение на печать. Оператор 50 присваивает значение знаменателю прогрессии, а оператор 60 подготавливает значение индекса $I = 2$ для вычисления второго элемента прогрессии.

Оператор 70 в цикле вычисляет значения элементов прогрессии и формирует массив этих значений, последовательно заполняя отведенные ячейки памяти.

---

* Имеется в виду версия языка БЕЙСИК, у которой минимальное значение индекса элементов массива равно 1.

Оператор 80 последовательно выводит на терминал элементы прогрессии, начиная со второго (первый элемент выведен ранее).

Оператор 90 увеличивает значение индекса для того, чтобы при вычислении очередного значения элемента прогрессии результат был помещен в следующую по порядку ячейку памяти. После вычисления 10 значений оператор 100 передает управление на окончание программы.

После завершения программы полученные значения элементов прогрессии сохраняются в ячейках массива A и при необходимости могут быть использованы в дальнейших вычислениях.

3.2. ПОСТРОЕНИЕ ЦИКЛОВ С ПОМОЩЬЮ ОПЕРАТОРОВ FOR и NEXT

Как было показано ранее, программы, содержащие циклы, могут быть организованы с помощью операторов передачи управления. Однако циклы являются настолько важной и распространенной конструкцией в программировании, что для их организации практически во всех языках программирования предусмотрены специальные операторы.

Операторы FOR и NEXT. Их используют для упрощения процедуры составления циклов и четкого выделения циклов в программе на языке БЕЙСИК.

Оператор FOR называют оператором заголовка цикла (или просто заголовком цикла). Он всегда предшествует повторяющейся группе операторов, составляющих так называемое *тело цикла*. Общая форма заголовка цикла имеет вид:

FOR v=e1 TO e2 STEP e3

где FOR — имя оператора (*для*); v — параметр цикла; e1 и e2 — начальное и конечное значения параметра цикла; TO — служебное слово (*до*); STEP — служебное слово (*шаг*); e3 — приращение, т. е. шаг изменения параметра цикла.

В случае, если v — параметр цикла целого типа и шаг изменения e3=1, оператор FOR допускает более компактную форму записи без указания приращения:

FOR v=e1 TO e2

За оператором FOR следуют операторы, составляющие тело цикла. Заканчивается цикл оператором NEXT. Общая форма оператора

NEXT v

где NEXT — имя оператора (*следующий*); v — параметр цикла.

Имя переменной, указанной в качестве параметра цикла в операторе NEXT, должно совпадать с именем переменной, указанной в качестве параметра цикла в заголовке цикла FOR.

При выполнении оператора NEXT изменяется значение параметра цикла (v = v+e3) и производится возврат на начало цикла. Опера-

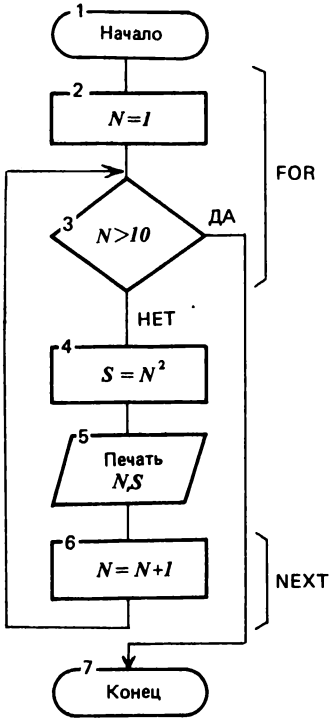


Рис. 3.3. Схема алгоритма циклического вычислительного процесса, организованного операторами FOR и NEXT

торы FOR и NEXT используются только в паре: FOR определяет начало цикла, а NEXT указывает конец тела цикла.

Цикл выполняется следующим образом. По оператору FOR сначала вычисляются характеристики (начальное и конечное значения, приращение (шаг)) и начальное значение присваивается параметру цикла. Затем производится проверка: не превышает ли начальное значение параметра цикла конечного значения. Если значение параметра цикла больше конечного значения по абсолютной величине, то производится выход из цикла, т. е. переход на оператор, следующий за оператором NEXT. Если значение параметра цикла меньше конечного значения, то далее выполняются операторы, составляющие тело цикла. По достижении оператора NEXT v происходит вычисление нового значения параметра цикла (v = v + e3) и переход на оператор FOR. Это значение снова сравнивается со значением e2. Цикл повторяется до тех пор, пока значение параметра цикла не станет строго больше v > e2 (при положительном приращении e3) или строго меньше v < e2 (при отрицательном приращении e3) конечного значения e2.

Следует иметь в виду, что начальное и конечное значения параметра цикла, а также шаг изменения параметра вычисляются и присваиваются только один раз при входе в цикл. Рассмотрим ряд примеров.

Пример 3.2. Составить программу вычисления и вывода на терминал квадратов всех чисел натурального ряда от 1 до 10. Схема алгоритма программы для этого примера приведена на рис. 3.3. Программа, составленная с использованием операторов цикла FOR и NEXT, будет иметь вид:

```
100 REM ВЫЧИСЛЕНИЕ КВАДРАТОВ ЧИСЕЛ
200 FOR N=1 TO 10
300 LET S=N^2
400 PRINT "N=";N,"S=";S
500 NEXT N
600 END
RUN

N= 1          S= 1
N= 2          S= 4
N= 3          S= 9
```

```
N=  4            S=  16
N=  5            S=  25
N=  6            S=  36
N=  7            S=  49
N=  8            S=  64
N=  9            S=  81
N=  10           S=  100

READY
```

Результаты работы этой программы полностью совпадают с результатами работы программы в примере 2.3. Как видно из примеров, программа с операторами FOR и NEXT компактнее программы, использующей оператор IF для организации цикла. Кроме того, для решения одной задачи можно составить несколько вариантов программ.

Пример 3.3. Вычислить и вывести на терминал таблицу значений функции $y = \sqrt{\sin x}$ для значений аргумента, изменяющихся от 0 до 0,5 рад с шагом 0,1 рад. Программу можно составить следующим образом:

```
5 REM ВЫЧИСЛЕНИЕ И ПЕЧАТЬ ТАБЛИЦЫ
10 FOR X=0 TO .5 STEP .1
20 LET Y=SQR(SIN(X))
30 PRINT "X=";X,"Y=";Y
40 NEXT X
50 END
RUN

X=  0            Y=  0
X=  .1           Y=  .315964
X=  .2           Y=  .445723
X=  .3           Y=  .543613
X=  .4           Y=  .624034
X=  .5           Y=  .692406

READY
```

Операторы 10 и 40 организуют цикл с параметром X, который будет принимать значения: 0; 0,1; 0,2; 0,3; 0,4; 0,5. Оператор 20 вычисляет значение функции Y. Оператор 30 организует вывод значений аргумента и функции. Оператор 40 увеличивает значения параметра цикла на шаг 0,1 рад и обеспечивает повторное выполнение операторов тела цикла, если условие выхода из цикла не удовлетворяется.

Операторы тела цикла будут выполнены 6 раз, в результате чего программа напечатает 6 строк.

Если в данном примере значение аргумента будет меняться в более широких пределах, например от 0 до 2,5, то под корнем может оказаться отрицательная величина. При этом машина не сможет вычислить соответствующее значение Y и выдаст сообщение об ошибке.

Правила использования операторов FOR и NEXT. При программировании циклов с помощью рассматриваемых операторов необходимо учитывать следующие правила.

Запрещается вход в цикл, минуя оператор FOR, т. е. передача

извне управления на операторы, составляющие тело цикла, недопустима. При такой передаче управления значения выражений e1, e2, e3 оказываются невычисленными. Если e1, e2, e3 — переменные или константы, то их значения не определены. Пример недопустимой передачи управления:

```
      ...
50 FOR J=K+2 TO 20
60 LET Y(J)=X(J)+5
70 NEXT J
80 LET A=A*3
90 IF A<=15 THEN 60
      ...
```

Здесь передача управления оператором 90 на внутренний оператор цикла 60 недопустима.

Извне управление передавать можно только на начало цикла, т. е. на оператор FOR.

Допускается выход из цикла в любое время. Например, во фрагменте программы

```
20 LET N=1
30 FOR I=1 TO 10
40 LET N=N*I
50 IF N>200 THEN 70
60 NEXT I
70 PRINT N,I
RUN

 720            6

READY
```

будет вычисляться значение $N = 1 \cdot 2 \cdot 3 \cdot 4 \ldots$ (т. е. факториал некоторого числа) до тех пор, пока значение $N <= 200$. При $N > 200$ с помощью оператора передачи управления 50 производится выход из цикла до естественного окончания цикла, организованного операторами FOR и NEXT. В результате выполнения этого фрагмента будет вычислено минимальное число I, факториал которого превышает 200.

Переопределение начального значения e1, *конечного значения* e2 *и шага изменения* e3 *параметра цикла в теле цикла не рекомендуется,* так как может привести к ошибкам.

Рекомендуется использование целых значений для параметров цикла с целью строгого соблюдения количества циклов. Переменная для обозначения параметра цикла также должна быть целой, например,

```
50 FOR J%=1% TO 10%
...
100 NEXT J%
```

*Сохраняется последнее значение параметра цикла при выходе из цикла**.

Построение графиков. Оператором цикла FOR и стандартной функцией TAB(X) удобно пользоваться при выводе на печать графиков

* В некоторых версиях языка это значение на 1 больше последнего значения.

функций. Рассмотрим организацию вывода на терминал графика функции на следующих примерах.

Пример 3.4. Построить график функции $y = \sin x$. Программа имеет вид:

```
10 REM ПОСТРОЕНИЕ ГРАФИКА ФУНКЦИИ
20 PRINT "ГРАФИК ФУНКЦИИ Y=SIN(X)"
30 FOR X=0 TO 2*PI STEP PI/8
40 PRINT TAB(15+10*SIN(X));"*"
50 NEXT X \ REM КОНЕЦ ЦИКЛА
60 END
RUN

ГРАФИК ФУНКЦИИ Y=SIN(X)
                    *
                        *
                            *
                              *
                               *
                              *
                            *
                        *
                    *
                *
            *
        *
        *
        *
            *
                *

READY
```

Здесь ось X направлена вертикально и проходит по 15-й позиции печати. Шаг по оси X определяется расстоянием между строками. Отклонения точек графика от оси X, соответствующие максимальному и минимальному значениям функции, составляют 10 позиций.

Как видно из примера 3.4, для построения графика функции необходимо провести предварительные расчеты:

найти максимальное и минимальное значения функции в заданном диапазоне изменения ее аргумента;

произвести масштабирование функции относительно ее минимального и максимального значения для того, чтобы точки графика не вышли за пределы формата листа (экрана);

определить позицию, соответствующую оси абсцисс;

выбрать шаг изменения аргумента. Другие, более точные приемы построения графических изображений рассмотрены в параграфе 4.4.

Рассмотрим пример, когда точки графика отмечаются числовыми значениями функции в этих точках.

Пример 3.5. Построить график функции $y = x^2$. Аргумент X меняется в диапазоне от — 6 до 6 с шагом 1:

```
10 REM ПОСТРОЕНИЕ ГРАФИКА ФУНКЦИИ
20 FOR X=-6 TO 6 \ PRINT TAB(X*X);X*X \ NEXT X
RUN
                                                  36
                                          25
                                  16
                          9
                  4
          1
  0
          1
                  4
                          9
                                  16
                                          25
                                                  36

READY
```

Ось X расположена перпендикулярно строкам на странице и проходит по нулевой позиции печати. Строка программы с номером 20 содержит три оператора.

Сложные (вложенные) циклы. Все рассмотренные выше примеры циклических программ содержали так называемые простые циклы. *Сложным* называют цикл, содержащий несколько вложенных один в другой простых циклов. В сложном цикле различают внешний и внутренние циклы. *Глубиной* сложного цикла называется количество вложенных друг в друга циклов. Сложные циклы можно организовать с помощью операторов FOR и NEXT.

Максимальное количество вложенных циклов в зависимости от системы, составляет 8—20. Структуру сложного цикла глубиной 3 можно представить в виде схемы, представленной на рис. 3.4. На этой схеме тело цикла с параметром V1 содержит два вложенных один в другой цикла; тело цикла с параметром V2 включает в себя цикл с параметром V3. При каждом повторении внешнего цикла внутренний цикл выполняется полностью, т. е. заданное число раз.

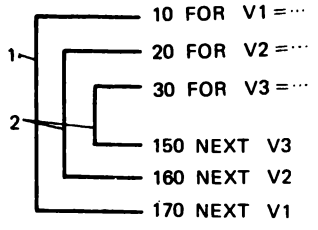


Рис. 3.4. Структура сложного цикла:
1 — внешний цикл; 2 — внутренние циклы

Пример 3.6. Определить сумму факториалов чисел от 1 до 5, т. е. $S=1!+2!+3!+4!+5!$ или $S=1+1\cdot2+1\cdot2\cdot3+1\cdot2\cdot3\cdot4+1\cdot2\cdot3\cdot4\cdot5$. Краткая математическая запись этого выражения:

$$S=\sum_{i=1}^{5} i!,$$

где Σ — знак суммы.

Эту задачу можно решить, используя сложный цикл. Во внутреннем цикле будем организовывать вычисление факториала по рекуррентной формуле

$$F = F * I$$

а во внешнем цикле — суммирование по рекуррентной формуле

$$S = S + F$$

Начальное значение F должно быть равно 1 и задаваться непосредственно перед внутренним циклом, а начальное значение S — равно 0 и задаваться до входа во внешний цикл. Программа будет иметь вид:

```
10 REM ВЫЧИСЛЕНИЕ СУММЫ ФАКТОРИАЛОВ
20 LET S=0
30 FOR I=1 TO 5
40 LET F=1
50 FOR J=1 TO I
60 LET F=F*J
70 NEXT J
80 LET S=S+F
90 NEXT I
100 PRINT "S=";S
110 END
RUN

S= 153

READY
```

Изменение параметров внешнего и внутреннего циклов приведено ниже.

I ... 1		2	3	4	5
J ... 1	1, 2	1, 2, 3	1, 2, 3, 4	1, 2, 3, 4, 5	

При организации сложных циклов необходимо учитывать следующее:

1. Все правила, сформулированные для простого цикла, должны соблюдаться и при организации сложных циклов.

2. Вход их внешнего цикла во внутренний, минуя оператор начала внутреннего цикла FOR, запрещен.

3. Имена параметров для циклов, вложенных один в другой, должны быть разными. Имена параметров для циклов, последовательно расположенных один за другим, могут быть одинаковыми.

4. Внутренний цикл должен полностью входить в тело внешнего цикла. Перекрытие (пересечение) циклов недопустимо.

Схема правильной организации циклов показана на рис. 3.5. В цикл с параметром A вложены три последовательных цикла с параметрами B, C, B. Цикл с параметром C в свою очередь является сложным: в него вложен цикл с параметром D. Все правила, перечисленные выше, соблюдаются.

Схема недопустимой организации циклов приведена на рис. 3.6. В первой группе циклов нарушено правило 4: циклы с параметрами A и B пересекаются. Во второй группе не соблюдено правило 3: внешний и внутренний циклы используют один и тот же параметр X. При

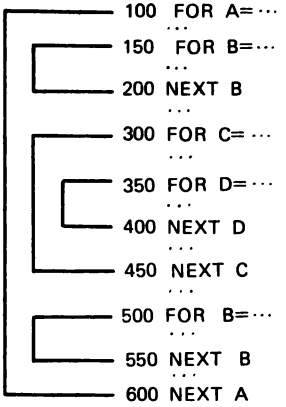


Рис. 3.5. Схема правильной организации циклов

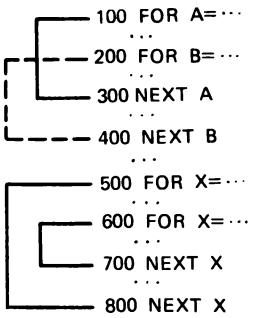


Рис. 3.6. Пример недопустимой организации циклов

такой ошибке внутренний цикл нарушает правильное выполнение внешнего цикла, изменяя значение его параметра.

Рассмотрим пример программы, выполняющей обработку числовых массивов.

Пример 3.7. Используя заданный двумерный массив **A**, имеющий 3 строки и 4 столбца, сформировать одномерный массив $\mathbf{B} = (b_1, b_2, b_3)$, компоненты которого определяются из соотношений

$$b_1 = a_{11} + a_{12} + a_{13} + a_{14};$$
$$b_2 = a_{21} + a_{22} + a_{23} + a_{24};$$
$$b_3 = a_{31} + a_{32} + a_{33} + a_{34},$$

т. е.

$$b_i = a_{i1} + a_{i2} + a_{i3} + a_{i4} \quad (i = 1, 2, 3) \text{ или}$$

$$b_i = \sum_{j=1}^{4} a_{ij}.$$

Для удобства составления программы не будем использовать нулевые значения индексов и опишем массивы A и B, указав в качестве максимальных значений индексов числа 3 и 4. При этом для массива A будет зарезервировано место для 20 элементов, а для B — для 4 элементов, хотя использоваться в программе будут 12 элементов массива A и 3 элемента массива B.

В программе необходимо предусмотреть следующие части:

1) формирование блока данных, содержащего значения элементов двумерного массива (матрицы) A;

2) ввод массива A (чтение значений элементов A(I, J) массива из блока данных);

3) вычисление элементов B(I) одномерного массива (вектора) B;

4) печать элементов массива B.

Необходимо также учитывать, что параметр внутреннего цикла должен пробегать все свои значения при одном конкретном значении параметра внешнего цикла. Иными словами, параметр внутреннего цикла должен изменяться быстрее, чем параметр внешнего цикла. В данном примере внутренний цикл следует организовать по параметру J, а внешний — по параметру I.

Ниже приведена программа, выполняющая перечисленные действия:

```
5 REM РАБОТА С ЧИСЛОВЫМИ МАССИВАМИ
10 DIM A(3,4),B(3)
20 REM ЗНАЧЕНИЯ ЭЛЕМЕНТОВ МАТРИЦЫ A
30 DATA 2.8, -4.9, 96, -1.2E3
40 DATA -8.6, 2.8, -6.45, 73
50 DATA 5.3E-2, 3.6, 10, 0.25
60 REM ВВОД МАТРИЦЫ A
70 FOR I=1 TO 3
80 FOR J=1 TO 4
90 READ A(I,J)
100 NEXT J
110 NEXT I
115 REM ВЫЧИСЛЕНИЕ ЭЛЕМЕНТОВ ВЕКТОРА B
120 FOR I=1 TO 3
130 LET S=0
140 FOR J=1 TO 4
150 LET S=S+A(I,J)
160 NEXT J
170 LET B(I)=S
180 NEXT I
185 REM ПЕЧАТЬ МАССИВА B
190 FOR I=1 TO 3
200 PRINT "B";I;"=";B(I)
210 NEXT I
220 END
RUN

B 1 =-1106.1
B 2 = 60.75
B 3 = 13.903

READY
```

В приведенной программе ввод элементов массива A выполняется с помощью сложного цикла (операторы 70—110), обеспечивающего построчный ввод массива. Вычисление элементов массива B выполняется также с использованием сложного цикла (операторы 120—180): внешний цикл с параметром I задает номер строки массива A, сумма элементов которой равна I-му элементу массива B; тело этого цикла содержит внутренний цикл с параметром J, вычисляющий сумму элементов I-й строки массива A. Для печати элементов массива B организован простой цикл с параметром I (операторы 190—210).

Следует отметить, что если k — число повторений внешнего цикла, а l — число повторений внутреннего цикла, то общее число повторений

операторов внутреннего цикла в программе из двух вложенных циклов будет $m = kl$. Так, в рассмотренной программе цикл вычисления (операторы 140—160) будет повторен $m = 3 \cdot 4 = 12$ раз.

Можно составить более краткую программу, совместив ввод элементов массива A, вычисление и печать элементов массива B. В этом случае операторы 60—220 нужно заменить на следующие:

```
60 REM ВВОД,ВЫЧИСЛЕНИЕ,ПЕЧАТЬ
70 FOR I=1 TO 3
80 LET S=0
90 FOR J=1 TO 4
100 READ A(I,J)
110 LET S=S+A(I,J)
120 NEXT J
130 LET B(I)=S
140 PRINT "B";I;"=";B(I)
150 NEXT I
160 END
```

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Что такое массив и как могут быть организованы массивы в языке БЕЙСИК?
2. Какой оператор служит для описания массивов, его общая форма?
3. Как размещаются массивы в памяти ЭВМ?
4. Какие операторы используются для организации программ с циклами?
5. Какие правила следует соблюдать при составлении программ с циклами с помощью операторов FOR и NEXT?
6. Как пользоваться стандартной функцией TAB (X) для построения графиков функций?
7. Что такое сложный цикл?
8. Как вычислить число повторений внутреннего цикла?

Глава 4

ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ЯЗЫКА «БЕЙСИК»

4.1. РАБОТА СО СТРОКОВЫМИ ДАННЫМИ

В языке БЕЙСИК можно использовать строковые константы и переменные, которые иначе называют *строковыми данными (строками символов)* (см. параграфы 1.2 и 1.3).

Над строковыми данными можно выполнять различные операции: присваивания, сцепления, отношения.

Операция присваивания. Строковой переменной можно присваивать значения строковой константы или значение другой строковой переменной. Операция присваивания для строковых данных выполняется аналогично этой операции для числовых данных и осуществляется с помощью оператора присваивания LET. Например.

```
40 LET A¤="BASIC"
50 LET B¤=A¤
60 LET C¤="РЕЗУЛЬТАТЫ"
```

Операция сцепления. При обработке строковых данных часто возникает необходимость их объединения для образования новой строки. Для этого предназначена операция сцепления. Ее обозначают знаком «+» или знаком & (амперсанд). Эта операция указывает, что два операнда соединяются вместе и образуют новую строку символов. При этом за последним символом левого операнда сразу же следует первый символ правого операнда. Длина полученной строки равна сумме длин операндов, но не должна превышать 255 символов. Операндами операции сцепления могут быть строковые константы и переменные. Например,

"ABCD" + "LM" эквивалентно "ABCDLM"

"157"& "КГ" эквивалентно "157 КГ"

или как в следующем фрагменте:

```
10 LET A¤="СИМ"
20 LET B¤="ВОЛ"
30 LET C¤=A¤+B¤
40 LET D¤=A¤+B¤+"ИКА"
```

В результате выполнения этого фрагмента переменная C¤ примет значение СИМВОЛ, а D¤ — СИМВОЛИКА.

Под *строковым выражением* в языке БЕЙСИК понимается любая

последовательность строковых констант, переменных и функций, соединенных с помощью операции сцепления «+». В рассмотренном выше примере во всех операторах справа от знака присваивания находятся строковые выражения.

Операции отношения. При программировании вычислительного процесса с разветвлением необходима проверка условия, от которого зависит дальнейший ход вычислений (см. параграф 2.5). Эта проверка над числовыми данными осуществляется с помощью операций отношения (=, <>, >, >=, <, <=). Эти же операции можно применить и к строковым данным, например:

условие "A" = "A" выполнено, т. е. результат выполнения операции отношения (=) есть *«истина»*;

условие "A" = "B" не выполнено, т. е. результат выполнения операции отношения (=) есть *«ложь»*;

условие "A" <> "B" выполнено;

условие C¤ = D¤ будет выполнено, если значение переменной C¤ равно значению переменной D¤ и не выполнено в противном случае.

В случае использования операций равно (=) и не равно (<>) результат выполнения условия очевиден и определяется совпадением или несовпадением правой и левой частей условия. Рассмотрим, как определяется истинность условия, содержащего сравнение строковых данных, когда в качестве операций отношения используются операции >, >=, <, <=.

Каждому символу языка БЕЙСИК предписывается некоторое числовое значение (код), которое определяется местом символа в машинном алфавите. Чем раньше он появляется в алфавите, тем меньшее числовое значение ему предписано. В машинном алфавите сначала идут некоторые специальные символы и знаки операций, затем цифры, потом латинские и русские буквы. Самое меньшее числовое значение предписано символу пробела. Например, числовое значение A меньше числового значения B, числовые значения A и B меньше числового значения C и т. д. Таблица числовых кодов символов приведена в приложении 7. Числовые значения, предписанные каждому символу, могут быть определены с помощью специальной функции ASC, о которой будет сказано ниже. При выполнении операций отношения и определении выполнения условия символы сравниваются с учетом их числовых значений. Например, условие "A" < "B" выполнено; условие "C" >= "F" не выполнено.

При сравнении двух строк, состоящих из нескольких символов, их длины предварительно выравниваются путем приписывания справа к более короткой строке недостающего числа пробелов. Затем производится посимвольное сравнение строк последовательно слева направо с учетом числовых значений, предписанных каждому символу. При различных символах строка 1 считается меньше строки 2, если сравниваемый символ строки 1 находится в алфавите раньше, чем соответствующий символ строки 2.

Например, условие "ABCD"<"ABCD" выполнено, так как символ B строки 1 в алфавите стоит раньше, чем символ C строки 2.

Рассмотрим следующий фрагмент программы:

```
10 LET A¤="STROKA"
20 LET B¤="SLOVO"
30 IF A¤>B¤ THEN 60
40 LET X=0
50 GO TO 70
60 LET X=1
70 ...
```

Здесь в операторе 30 условие A¤ > B¤ выполнено, так как первый символ строк A¤ и B¤ один и тот же, а числовое значение символа T строки A¤ больше числового значения символа L строки B¤. Следовательно, в результате выполнения оператора 30 управление будет передано оператору 60 и X получит значение, равное 1. Если бы строка A имела значение, например, BUKVA, то условие в операторе 30 не выполнится и X получит значение 0.

Стандартные строковые функции*. Над строками символов можно выполнять различные стандартные действия. Они выполняются с помощью специальных строковых функций, которые позволяют программе выбирать часть строки, определять ее длину, формировать заданную строку или значение, выполнять поиск подстроки в строке большего размера и другие полезные операции. Строковые функции можно включать в строковые выражения. Рассмотрим некоторые из этих функций. Их аргументами могут быть строковые или числовые данные.

Если имя функции оканчивается символом ¤, то в результате ее выполнения создается строка символов; если символ ¤ отсутствует в имени функции, результатом будет целое десятичное значение.

Функция LEN определяет длину строки, т. е. выдает в качестве результата количество символов в символьной строке. Формат функции

LEN (*строка*)

где *строка* — строковая константа или переменная.

Ниже показан пример применения этой функции:

```
10 LET A¤="ПРОГРАММА"
20 PRINT LEN(A¤)
RUN

 9

READY
```

Функция POS** осуществляет поиск подстрок в строке. Подстрока является ее частью. Формат функции

POS (*строка* 1, *строка* 2, e)

где *строка 1* — строка, в которой осуществляется поиск; *строка 2* — подстрока; e — позиция символа (десятичное число), с которого начинается поиск.

---

* Ниже рассматриваются стандартные строковые функции БЕЙСИК системы ДВК. Для БЕЙСИК системы MSX форма некоторых стандартных функций несколько отличается от приводимых в данном параграфе.

** В языке БЕЙСИК—MSX с этой целью используется функция INSTR.

Функция POS указывает позицию, с которой *строка 2* встречается впервые в *строке 1*. Она равна 0, если *строка 2* не содержится *в строке 1*. Например,

```
20 LET A¤="ПРОГРАММИРОВАНИЕ"
30 LET B¤="РО"
40 LET L=POS(A¤,B¤,1)
50 LET M=POS(A¤,B¤,4)
60 LET N=POS(A¤,"МА",1)
70 PRINT "L=";L,"M=";M,"N=";N
80 END
RUN

L= 2          M= 10          N= 0

READY
```

Функция SEG¤* выделяет подстроку из символьной строки. Формат функции

SEG ¤ (*строка*, e1, e2)

где *строка* — строка символов; e1, e2 — позиции первого и последнего выделяемых символов.

В результате выполнения функции SEG ¤ из строки выделяется подстрока, начиная с символа, находящегося на позиции e1 и кончая символом, находящимся на позиции e2. Например,

```
20 LET A¤="ПОДПРОГРАММА"
30 LET B¤=SEG¤(A¤,4,12)
40 PRINT B¤,SEG¤(B¤,4,8)
RUN

ПРОГРАММА      ГРАММ

READY
```

Если значение e1 оказывается больше значения e2, или длины строки, то результатом будет пустая строка. Если e2 больше длины строки, то результатом будет исходная строка. Если e1 равно e2, то будет выделен один символ.

Функция TRM¤ исключает в заданной строке конечные пробелы. Формат функции

TRM ¤ (*строка*)

Например,

```
20 LET B¤="СИМ  "
30 LET C¤="ВОЛ"
40 PRINT B¤+C¤,TRM¤(B¤)+C¤
RUN
СИМ  ВОЛ       СИМВОЛ

READY
```

---

* В языке БЕЙСИК — MSX с этой целью используется функция MID¤.

Функция ASC выдает десятичный код символа в машинном алфавите. Формат функции

ASC (*символ*)

где *символ* — символ языка БЕЙСИК.

В качестве аргумента функции должен быть только один символ. Например,

```
PRINT ASC("D")
68

PRINT ASC("Б")
98
```

Функция CHR¤ выдает символ по его коду в машинном алфавите. Формат функции

CHR ¤ (*n*)

где *n* — десятичный код символа в машинном алфавите в пределах от 0 до 127 (номера в пределах от 0 до 31 соответствуют управляющим символам клавиатуры и не входят в алфавит языка БЕЙСИК). Например,

```
PRINT CHR¤(65)
A

PRINT CHR¤(42)
*

PRINT CHR¤(32)
```

печать пробела (т. е. пустая позиция)

В разных версиях языка БЕЙСИК строковые функции и их аргументы могут несколько отличаться по написанию и содержанию от рассмотренных выше, например в версии БЕЙСИК — ПЛЮС. Некоторые версии языка БЕЙСИК вообще могут не содержать строковых функций, например версия для микроЭВМ «Электроника-60».

Ввод — вывод строковых данных. В основном он аналогичен вводу — выводу числовых данных, хотя и имеет некоторую специфику. Для ввода — вывода строковых данных используются известные операторы READ, DATA, RESTORE, PRINT. Наряду с оператором INPUT при вводе строковых данных используется оператор LINPUT. Формат оператора

LINPUT (*стрпер*1, *стрпер* 2, ..., *стрпер* N)

где *стрпер* — строковая переменная.

В списке оператора LINPUT строковые переменные отделяются друг от друга запятыми. Встретив оператор LINPUT, БЕЙСИК-система печатает знак «?» и ждет ввода данных для первой переменной списка. Значением переменной является строка информации (с терминала), включая начальные, конечные и промежуточные пробелы, символы пунктуации и кавычки. При нажатии клавиши ⟨ВК⟩ эта строка вводится в ЭВМ и присваивается первой строковой переменной из

списка оператора LINPUT. БЕЙСИК-система снова печатает знак «?» и ждет ввода с терминала очередной строки. После ввода значения для последней переменной списка программа продолжит исполняться с оператора, следующего за оператором LINPUT. Например,

```
20 LINPUT A¤,B¤
30 PRINT "A¤=";A¤,"B¤=";B¤
RUN

? ОПЕРАТОР ВВ,"L"
? "ПРОГРАММА 1"
A¤=ОПЕРАТОР ВВ,"L"          B¤="ПРОГРАММА 1"

READY
```

Оператор INPUT может применяться одновременно для ввода строковых и числовых данных.

При использовании оператора READ совместно с оператором DATA строковую константу нужно заключить в кавычки, если в ней есть запятая или пробел. Она может быть без ковычек, если не содержит этих символов. Например,

```
10 DATA "ВВОД ВЫВОД",ДАННЫЕ,"SIN,COS"
30 READ A¤,B¤,C1¤
40 PRINT A¤
50 PRINT B¤,C1¤
60 END
RUN

ВВОД ВЫВОД
ДАННЫЕ          SIN,COS

READY
```

В языке БЕЙСИК допускается чтение числовой константы в строковую переменную, например,

```
10 DATA 16.4
20 READ A¤
```

В этом случае числовая константа в дальнейшем будет использоваться как символьная. Попытка считать строковую константу в числовую переменную вызовет сообщение об ошибке.

Оператор RESTORE используется одинаково для строковых и числовых констант.

Строковые данные могут быть объединены в массив. Например, фрагмент программы

```
10 DIM T¤(5)
20 FOR I=1 TO 5
30 LINPUT T¤(I)
40 NEXT I
```

организует ввод элементов массива строковых данных с терминала. В данном случае нужно ввести пять строк.

4.2. НЕСТАНДАРТНЫЕ ФУНКЦИИ, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ

Кроме стандартных функций в языке БЕЙСИК возможно употребление функций, составленных самим пользователем. Если в программе необходимо несколько раз вычислять одно и то же выражение при различных значениях параметров, то с целью экономии памяти машины и упрощения программы целесообразно создать свою функцию, присвоив ей некоторое имя. В отличие от стандартной функции такую функцию называют *нестандартной*. Она дает возможность обращаться к ней по имени так же, как и к стандартной математической функции. Нестандартная функция определяется оператором DEF, который присваивает ей задаваемое пользователем имя. Формат оператора

DEF FNv (x1, x2, ..., xN) = e

где DEF — имя оператора (*определение*); FNv — имя функции, определяемой пользователем (v — латинская буква, выбираемая программистом); x1, x2, ..., xN — список формальных аргументов (для ДВК количество формальных аргументов не более пяти); e — арифметическое выражение.

Например,

20 DEF FNA(X, Y) = X∧2 + Y∧2

Имя нестандартной функции всегда составляется из трех латинских букв, первые две из которых зафиксированы — FN, а третья выбирается пользователем. Формальными аргументами x1, x2, ..., xN могут быть только имена простых (неиндексированных) переменных, разделенных запятыми. Как правило, формальные аргументы входят в арифметическое выражение e. В него могут входить также другие переменные и константы.

Арифметическое выражение e указывает, какие действия и в какой последовательности надо выполнить с формальными аргументами и другими входящими в него величинами. Так, в приведенном выше примере в операторе 20 указано, что некоторую величину (безразлично какую, т. е. формальную) X нужно возвести в квадрат и сложить с квадратом другой формальной величины Y, т. е. задан шаблон для проведения аналогичных вычислений. После того как нестандартная функция описана в программе оператором DEF, ее можно использовать в этой программе так же, как и стандартные функции, т. е. обращаться к ней в различных операторах языка БЕЙСИК.

Пример 4.1. Составить программу вычисления величины

$$z = \frac{3a^2 + \cos a - 3b^2 - \cos b}{[3(a+b)^2 + \cos(a+b)]^3}.$$

Здесь многократно используется формула, которую можно записать как $y(x) = 3x^2 + \cos x$.

В программе эту формулу целесообразно определить как функцию с помощью оператора DEF. Программа имеет вид:

```
10 REM ОПРЕДЕЛЕНИЕ ФУНКЦИИ
20 DEF FNY(X)=3*X^2+COS(X)
30 REM ВВОД ЗНАЧЕНИЙ ДЛЯ A И B
40 INPUT A,B
50 REM ВЫЧИСЛЕНИЕ Z
60 LET Z=(FNY(A)-FNY(B))/FNY(A+B)^3
70 PRINT "Z=";Z
80 END
RUN

? 0.5
? 14
Z=-2.34119E-06

READY
```

Аргументы функции FNY, указанные в операторе 60, называются *фактическими аргументами*. Они задают конкретные (фактические) числовые значения, которые подставляются на место формального аргумента в операторе 20. Отметим ряд особенностей, которые следует учитывать при программировании нестандартных функций.

Формальные аргументы используются лишь для обозначения аргументов и указания их вхождения в формулу вычисления значения функции. Например, во фрагменте

```
10 DEF FNA(X,Y)=SQR(X^2+Y^2)
...
40 DEF FNB(X)=SIN(X)/X
...
70 LET Z=Y+B*3
```

аргументы X в определении функций FNA и FNB — это не связанные между собой величины. В вычислительной машине им будут отведены разные ячейки оперативной памяти. Точно так же формальный аргумент Y в операторе 10 не имеет никакого отношения к простой переменной с аналогичным именем (оператор 70) в программе на языке БЕЙСИК.

Фактическими аргументами могут быть: константы, переменные (простые и индексированные), выражения.

Обращение к нестандартной функции происходит лишь в тот момент, когда в качестве операнда в формуле выполняемого оператора встречается имя нестандартной функции с фактическими аргументами. При обращении к функции сначала вычисляются значения всех фактических аргументов, далее они подставляются вместо соответствующих формальных аргументов в описание функции и вычисляется значение выражения, записанного в правой части определения функции. Это значение и является значением функции.

Функции, определяемые пользователем, могут быть целочисленными, вещественными или строковыми. Функция объявляется целочисленной с помощью символа % или строковой с помощью символа ¤ после имени функции. Значением целочисленной функции является целое число, значением строковой функции — строка символов. Если знаки % и ¤

отсутствуют, то функция — действительного типа, т. е. результат — вещественное число.

Примеры употребления функций, определенных пользователем, приведены ниже

```
10 DEF FNI%(X,Y,K%)=X*K%+Y*10*K%
10 DEF FNS¤(A¤,L)=SEG¤(A¤,1,L)+"A"
10 DEF FNR(X)=SIN(X)/X
```

При таком определении значением функции FNI% будет целое число; FNS¤— строка символов; FNR — вещественное число.

Пользователь должен следить за тем, чтобы при определении функций не возникали недопустимые преобразования типов данных, т. е. преобразования числовых данных в строковые и наоборот.

Функция любого типа может зависеть от формальных аргументов различных типов — целочисленных вещественных или строковых. При обращении к ней количество фактических аргументов и их тип должны соответствовать количеству и типу формальных аргументов в определении функции. Нарушение этого требования приводит к ошибке. Например, в результате выполнения операторов

```
10 DEF FNA(X)=2*X/SQR(X)
20 PRINT FNA(5,2)
```

будет выведено сообщение об ошибке в аргументах:

```
? ARGUMENT ERROR AT LINE 20
```

Оператор DEF — невыполняемый оператор и поэтому рекомендуется все операторы DEF размещать вместе с другими невыполняемыми операторами в начале программы. Он может использоваться только в программном режиме.

4.3. ПОДПРОГРАММЫ

Организация подпрограммы. Специальным образом оформленная группа операторов, к которым можно обращаться из разных точек программы, называется *подпрограммой*. Ее используют тогда, когда часто повторяемая вычислительная процедура не укладывается в одну формулу и не может быть описана в операторе DEF.

Особенностью в оформлении подпрограммы является то, что ее действие должно завершиться специальным оператором возврата из подпрограммы.

Общая форма оператора возврата

RETURN (*возврат*)

где RETURN — имя оператора (*возврат*).

Оператор не имеет аргументов.

В качестве первого оператора подпрограммы обычно используют невыполняемый оператор REM (*комментарий*).

Рассмотрим пример подпрограммы, вычисляющей факториал:

```
1000 REM ВЫЧИСЛЕНИЕ ФАКТОРИАЛА
1010 REM  K-ВХОДНОЙ ПАРАМЕТР
1020 REM  F-РЕЗУЛЬТАТ
1030 LET F=1
1040 FOR I=1 TO K
1050 LET F=F*I
1060 NEXT I
1070 RETURN
```

Подобно оператору DEF, величины K и F, записанные в подпрограмме, являются формальными аргументами. Конкретные значения, которые они будут получать, будут фактическими аргументами. Перед выполнением подпрограммы значение входного параметра K должно быть определено.

Для обращения к подпрограмме используют оператор GOSUB (*перейти к подпрограмме*). Формат оператора:

GOSUB m1

где GOSUB — имя оператора; m1 — номер первого оператора подпрограммы.

Оператор GOSUB запоминает то место, из которого происходит обращение к подпрограмме, и передает управление оператору с номером m1. После выполнения подпрограммы оператор RETURN возвращает управление оператору, следующему за оператором GOSUB.

Подпрограмма может располагаться в любом месте программы, однако ее удобнее размещать в конце программы после оператора STOP и перед оператором END.

При использовании подпрограммы в языке БЕЙСИК программист должен сам заботиться о присвоении необходимых значений входным параметрам подпрограммы и о сохранении результатов ее выполнения, иначе они будут испорчены при повторных обращениях к ней.

Пример 4.2. Составить программу для вычисления числа сочетаний из n элементов по m $(n \geqslant m)$.

Число сочетаний вычисляется по формуле

$$C_n^m = \frac{n!}{m!(n-m)!}.$$

Вычисление факториала оформлено ранее в виде подпрограммы, у которой входным параметром является переменная K, а результатом — переменная F. Перед каждым обращением к подпрограмме необходимо задавать значение входного параметра, а после возврата из нее запоминать («спасать») вычисленный результат.

Введем следующие обозначения: $n! = N1$; $m! = M1$; $C_n^m = C$; $n = N$; $m = M$.

Ниже приведена программа, вычисляющая число сочетаний:

```
5 REM ОСНОВНАЯ ПРОГРАММА (ВЫЧИСЛЕНИЕ C)
10 INPUT N M
20 IF N>=M THEN 50
```

```
30 PRINT " НАРУШЕНО УСЛОВИЕ N>=M. ПОВТОРИТЕ ВВОД"
40 GO TO 10
50 REM ВЫЧИСЛИТЬ N!
60 LET K=N
70 GOSUB 1000
80 LET N1=F
90 REM ВЫЧИСЛИТЬ M!
100 LET K=M
110 GOSUB 1000
120 LET M1=F
130 REM ВЫЧИСЛИТЬ (M-N)!
140 LET K=N-M
150 GOSUB 1000
160 REM ВЫЧИСЛИТЬ ЧИСЛО СОЧЕТАНИЙ
170 LET C=N1/(M1*F)
180 PRINT "N=";N,"M=";M,"C=";C
190 STOP
1000 REM ПОДПРОГРАММА (ВЫЧИСЛЕНИЕ ФАКТОРИАЛА)
1010 REM   K-ВХОДНОЙ ПАРАМЕТР
1020 REM   F-РЕЗУЛЬТАТ
1030 LET F=1
1040 FOR I=1 TO K
1050 LET F=F*I
1060 NEXT I
1070 RETURN
1080 END
RUN
? 14,4
N= 14          M= 4           C= 1001

STOP AT LINE 190

READY
RUN

? 10,3
N= 10          M= 3           C=120

STOP AT LINE 190

READY
```

Операторы GOSUB 1000 передают управление невыполняемому оператору REM, который не влияет на выполнение программы. Вместо операторов GOSUB 1000 можно было бы использовать GOSUB 1030.

Операторы 60, 100, 140 передают значения фактических аргументов N, M, N—M в подпрограмму.

Операторы 80 и 120 запоминают («спасают») вычисленные значения n! и m!. Например, если бы не было оператора 80, то после вторичного вызова подпрограммы по оператору 110 значение n!, которое находилось в ячейке F, было бы затерто значением m!.

Вложенные подпрограммы. Подпрограммы могут содержать обращения к другим подпрограммам. При такой записи внутренняя подпро-

грамма считается вложенной. Пусть требуется считать данные, вычислить их по разным формулам и напечатать полученные результаты. Вычисления по формулам и печать результатов оформим как подпрограммы.

В приведенной ниже программе оператор GOSUB в строке 20 вызывает подпрограмму вычислений, начинающуюся со строки 100, которая, в свою очередь, вызывает вложенную подпрограмму печати в строках 120 и 150. Заметим, что здесь один оператор RETURN обслуживает три оператора GOSUB:

```
5 REM ВЛОЖЕННЫЕ ПОДПРОГРАММЫ
10 DATA 2,4
15 READ A,B
20 GOSUB 100
25 STOP
100 LET X=A
115 LET Y=B
120 GOSUB 180
130 LET X=X^2
140 LET Y=Y^2
150 GOSUB 180
160 LET X=X+1/A
170 LET Y=Y-1/A
180 PRINT X,Y
190 RETURN
RUN

2              4
4              16
4.5            15.5


STOP AT LINE 25

READY
```

БЕЙСИК-система, встретив в программе оператор GOSUB, передает управление строке подпрограммы, заданной в этом операторе. Подпрограмма выполняется с заданной строки, пока не встретится оператор RETURN возврата из подпрограммы. БЕЙСИК-система организует таблицу адресов возврата. Всякий раз при выполнении оператора GOSUB БЕЙСИК-система помещает в таблицу адрес строки, следующей за этим оператором. Для версии БЕЙСИК-ОС ДВК эта таблица вмещает не более 20 адресов строк.

4.4. ФОРМИРОВАНИЕ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ

Искусственный прием построения графического изображения с помощью алфавитно-цифровых символов, имеющихся на клавиатуре ЭВМ и выводимых в нужное место экрана оператором PRINT с функцией TAB,

был рассмотрен в параграфе 3.2. Но использование алфавитно-цифрового представления информации не всегда удобно. В настоящее время все более широко применяют графическое представление информации.

Современные дисплеи могут работать не только в алфавитно-цифровом режиме, но и графическом, когда на экране можно представить произвольные графические объекты. Изображение высвечивается подобно изображению на экране телевизора. Каждая линия представляет собой совокупность светящихся точек. Так как этих точек достаточно много, глаз не различает каждую отдельную точку и линия представляется нам сплошной.

В большинстве дисплеев на рабочем поле экрана по горизонтали расположено 256, а по вертикали — 192 точки (в некоторых конструкциях дисплеев — 240 точек). Каждая точка имеет свои координаты. При этом начало координат помещено в левом верхнем углу экрана, ось абсцисс направлена слева направо, а ось ординат — сверху вниз. Таким образом, левый верхний угол экрана (рис. 4.1) имеет координаты (0, 0), а правый нижний угол (255, 191).

В зависимости от конструкции дисплея изображение может быть как двухцветным (например, черно-белым), так и цветным (иметь до 16 различных цветов).

Специальные средства (операторы и функции) языка БЕЙСИК позволяют работать с графической информацией.

Подготовка дисплея для работы в режиме графики. В этом режиме можно использовать следующие операторы.

Оператор SCREEN переключает режимы работы дисплея, в частности включает режим графики. Общая форма оператора

SCREEN ⟨*номер режима*⟩

где SCREEN — имя оператора (*экран*); ⟨*номер режима*⟩ — десятичная цифра, определяющая режим работы дисплея.

Если ⟨*номер режима*⟩ равен 0 или 1, то будет установлен алфавитно-цифровой режим, если ⟨*номер режима*⟩ равен 2 или 3, то — графический. При входе в графический режим экран дисплея разбивается на рабочее поле и рамку. Координатные оси не высвечиваются.

Оператор COLOR задает необходимый цвет изображения. Общая форма оператора

COLOR ⟨*номер цвета*⟩

где COLOR — имя оператора (*цвет*); ⟨*номер цвета*⟩ — десятичное число, соответствующее определенному цвету изображения.

Например, оператор 10 COLOR 1 означает, что изображение будет рисоваться белым цветом.

В зависимости от конструкции дисплея номера цветов могут различаться и одному и тому же цвету в разных конструкциях могут соответствовать различные числа.

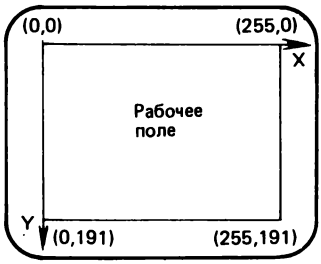


Рис. 4.1. Экран дисплея в графическом режиме

В программе оператор COLOR следует ставить перед входом в режим графики (перед оператором SCREEN).

Оператор CLS — очищает экран дисплея. Общая форма оператора

CLS

где CLS — имя оператора (*очистить экран*).

Функции оператора CLS может выполнять также оператор SCREEN.

Построение прямых линий и прямоугольников. Задача формирования графического изображения на экране дисплея представляет собой процесс последовательного применения операторов машинной графики. Одним из них является оператор LINE. Он служит для построения на экране прямых линий и прямоугольников. Основная форма записи оператора:

LINE (x1, y1) — (x2, y2), ⟨*номер цвета*⟩

где LINE — имя оператора (*линия*); x1, y1 — координаты начальной точки; x2, y2 — координаты конечной точки; ⟨*номер цвета*⟩ — десятичное число, указывающее номер цвета, которым вычерчивается объект.

Если параметр ⟨*номер цвета*⟩ не задан, то объект вычерчивается цветом, указанным в операторе COLOR.

По оператору LINE на экране дисплея вычерчивается отрезок прямой линии с координатами начальной (x1, y1) и конечной (x2, y2) точек отрезка. Если координаты начальной точки (x1, y1) отсутствуют, то за начальную точку принимается точка окончания предыдущих построений, либо точка с координатами (0, 0), если предыдущих построений не было.

Рассмотрим пример программы для построения линий:

```
10 REM ОТРЕЗКИ
20 SCREEN 2
30 CLS
40 LINE(20,80)-(80,80),1
50 LINE -(140,140),1
60 LINE(100,140)-(180,140),1
70 FOR I=1 TO 1000
80 NEXT I
```

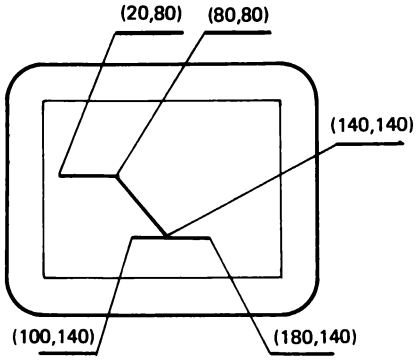


Рис. 4.2. Изображение линий на экране (к примеру)

Согласно этой программе будет построено изображение, состоящее из трех линий. Координаты точек на экране не изображаются (рис. 4.2). В операторе с номером 50 отсутствуют координаты начальной точки. В этом случае за начальную точку принимается окончание предыдущего построения, т. е. точка с координатами (80, 80).

Операторы 70 и 80, организующие пустой цикл, используются для того, чтобы задер-

жать на некоторое время изображение на экране. Это необходимо вследствие того, что выключение графического экрана происходит автоматически при завершении графических построений.

Оператор LINE может иметь и расширенную форму:

LINE (x1, y1) — (x2, y2), ⟨*номер цвета*⟩, ⟨*признак*⟩

где ⟨*признак*⟩ — параметр, указывающий вид фигуры, изображенной на экране.

Параметр ⟨*признак*⟩ может иметь значения: B — изображается прямоугольник; BF — закрашенный прямоугольник.

Введение в оператор LINE признаков B или BF не меняет содержания первых параметров. Они по-прежнему задают отрезок прямой, который теперь рассматривается как диагональ изображаемого прямоугольника. При этом сама диагональ на экране не изображается. Например,

```
10 REM СХЕМА
20 SCREEN 2
30 CLS
40 LINE(20,100)-(40,100),1
50 LINE(40,90)-(80,110),1,B
60 LINE(80,100)-(140,100),1
70 LINE(140,80)-(200,120),1,BF
80 FOR I=1 TO 2000
90 NEXT I
```

В этой программе оператор с номером 50 вычерчивает прямоугольник, а оператор 70 — закрашенный прямоугольник. Процесс составления программы для построения графического изображения можно несколько упростить, если требуемое изображение предварительно сформировать вручную на листе миллиметровой бумаги размером 256× ×192 мм.

Построение окружностей. Для построения окружностей используется оператор CIRCLE. Простейшая форма оператора

CIRCLE (x, y), ⟨*радиус*⟩, ⟨*номер цвета*⟩

где CIRCLE — имя оператора (*окружность*); x, y — координаты центра окружности; ⟨*радиус*⟩ — радиус окружности; ⟨*номер цвета*⟩ — десятичное число, указывающее номер цвета, так же как и в операторе LINE.

Рассмотрим пример программы для построения в цикле четырех окружностей радиуса R = 20 (рис. 4.3).

```
10 REM ОКРУЖНОСТИ
20 SCREEN 2
30 CLS
40 LET R=20
50 FOR L=60 TO 180 STEP 40
60 CIRCLE( L,80 ),R,1
70 CIRCLE( L,80 ),10,1
80 NEXT L
90 GO TO 90
```

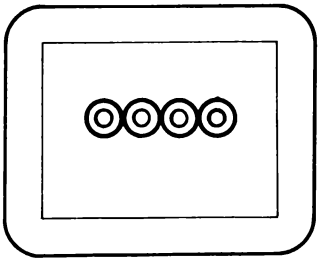


Рис. 4.3. Изображение окружностей (к примеру)

Центры окружностей находятся на одной линии (оператор 60). Внутри каждой из них помещены окружности меньшего радиуса (оператор 70).

Оператор 90, передающий управление самому себе, задерживает изображение на экране неопределенно долгое время.

Аргументы x1, y1, x2, y2 в операторе LINE и x, y, ⟨*радиус*⟩ в операторе CIRCLE могут быть не только константами и переменными, но и арифметическими выражениями. Результаты вычисления арифметических выражений могут принимать значения от —32768 до 32767, однако при этом на экране отображаются только те точки, координаты которых находятся в диапазоне

$$0 <= x <= 255,\ 0 <= y <= 191.$$

Построение изображений по точкам. Выше было рассмотрено построение графических изображений из элементарных геометрических объектов (прямых линий, прямоугольников, окружностей). В языке БЕЙСИК можно построить на экране отдельно взятую точку. Для этого служит оператор PSET. Общая форма оператора

PSET (x, y), ⟨*номер цвета*⟩

где PSET — имя оператора (*установка точки*); x, y — координаты точки; ⟨*номер цвета*⟩ — десятичное число, указывающее номер цвета точки на экране (задается так же, как и в операторах LINE и CIRCLE).

Оператор PSET строит на экране точку с указанными координатами. Аргументы x и y, задающие координаты точки, могут быть константами, переменными и арифметическими выражениями. Если координаты заданной точки находятся за пределами экрана, то оператор PSET не работает.

Пример 4.3. Составим программу, использующую оператор PSET:

```
10 REM ТОЧКИ
20 CLS
30 SCREEN 2
40 PSET(100,5),1
50 PSET(100,15),1
60 FOR X=1 TO 200
70 PSET(X,0),1
80 PSET(X,SQR(X)),1
90 NEXT X
100 FOR I=1 TO 1000
110 NEXT I
```

Эта программа строит сначала две точки с координатами (100, 5) и (100, 15) (операторы 40 и 50), а затем в цикле две линии по точкам (рис. 4.4). Одна из этих линий совпадает с осью x (оператор 70), а другая представляет собой график функции $y = \sqrt{x}$ (оператор 80).

Оператор PSET можно использовать для построения изображений по шаблону.

Пример 4.4. Пусть требуется изобразить на экране дисплея цифру 5 в форме, отличной от стандартной.

Составим шаблон, т. е. изображение этой цифры на бумаге в клетку (рис. 4.5). Закодируем заштрихованные участки единицами, а светлые — нулями. В результате получим таблицу

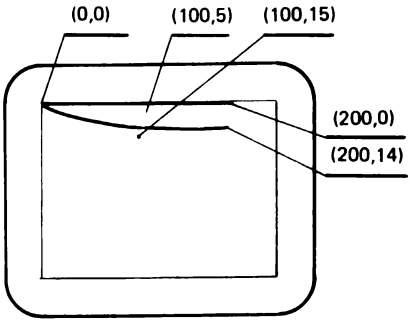


Рис. 4.4. Построение изображений по точкам

1	1	1
1	0	0
1	1	0
0	0	1
0	0	1
1	1	0

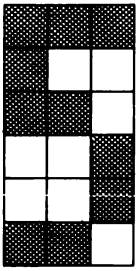


Рис. 4.5. Шаблон для цифры «пять»

имеющую 6 строк и 3 столбца. Координаты точек, которые должны будут высвечены на экране, будем определять в зависимости от координат единиц в таблице и того места экрана, на которое предполагается вывести требуемое изображение. Координатами единицы в таблице является номер строки и столбца, на пересечении которых находится единица.

Введем данную таблицу в ЭВМ, т. е. создадим блок данных из значений ее элементов. Далее, считывая по строкам значения из этой таблицы в некоторую переменную P и анализируя считанное значение, можно высветить точку на экране с помощью оператора PSET, если P = 1. В противном случае, если P = 0, ничего не высвечивается и выполняется переход к следующему элементу таблицы. Таким образом, мы сможем вывести изображение (цифру 5) в любое место экрана в требуемом масштабе.

Ниже приведена программа, выполняющая рассмотренные построения:

```
10 REM ШАБЛОН
20 CLS
30 SCREEN 2
40 DATA 1,1,1
```

```
50 DATA 1,0,0
60 DATA 1,1,0
70 DATA 0,0,1
80 DATA 0,0,1
90 DATA 1,1,0
100 FOR I=1 TO 6
110 FOR J=1 TO 3
120 READ P
130 IF P=0 THEN 170
140 PSET(J+50,I+50),1
150 PSET(J*3+100,I*3+50),1
160 PSET(J*10+150,I*10+50),1
170 NEXT J
180 NEXT I
190 FOR L=1 TO 1000
200 NEXT L
```

Здесь операторы 140, 150, 160 высвечивают три фигуры, изображающие цифру 5. Оператор 140 высвечивает в соответствии с шаблоном фигуру со смещением по осям x и y на 50 единиц, оператор 150 — фигуру со смещением по оси x на 100 единиц, а по оси y — на 50 единиц. Расстояния между точками, образующими фигуру, будут увеличены по сравнению с расстояниями в исходной фигуре в 3 раза. Фигура, построенная оператором 160, будет иметь смещение по оси x уже 150 единиц, а по оси y, как и ранее, — 50 единиц, и расстояния между точками будут увеличены по сравнению с расстояниями в исходной фигуре в 10 раз.

Аналогично можно вычерчивать любые фигуры по предварительно сформированным шаблонам.

4.5 ФОРМАТИРОВАННАЯ ПЕЧАТЬ РЕЗУЛЬТАТОВ

При выводе информации с помощью оператора PRINT БЕЙСИК-система сама выбирает форму выводимых данных. Программист имеет лишь минимальные возможности изменения этой формы. В ряде случаев пользователя может не удовлетворять такой стандартный вывод, а требуется заранее указать точное представление и расположение выходных данных. Для этого используется специальный оператор PRINT USING, организующий вывод значений числовых и строковых переменных по специальным форматам. Общая форма оператора

PRINT USING ⟨*форматная строка*⟩, ⟨*список*⟩

где PRINT USING — имя оператора (*печать по формату*); ⟨*форматная строка*⟩ — строка символов, определяющая представление выводимых данных; ⟨*список*⟩ — перечень элементов, подлежащих выводу.

По этому оператору значения элементов, указанных в списке, выводятся на терминал согласно формата, представленного форматной строкой. В списке могут находиться арифметические и строковые константы и переменные, а также арифметические и строковые перемен-

ные. Форматная строка определяет с точностью до позиции вид выводимой информации и указывает форму представления выводимых значений. Ею может быть строковая константа, строковая переменная, либо строковое выражение.

Форматная строка содержит любые символы языка БЕЙСИК. Среди них особое значение имеют специальные знаки управления форматом: ', \, #, ^, +, —, ., *, $, и некоторые буквы, например L, R, C. В форматной строке задаются поля, т. е. количество позиций и вид информации, которая будет напечатана в этих позициях для соответствующих элементов списка. Имеется два типа полей: числовые и строковые. Для каждого элемента списка в форматной строке соответственно должен быть указан свой формат. *Формат поля* — это условное обозначение, указывающее, в какие позиции и в каком виде будет выведена информация. Форматы полей составляются из знаков управления форматом. Например, если в списке находится переменная, имеющая числовое значение, то для нее в форматной строке должен быть указан формат числового поля, а если в списке находится строковая константа или переменная, то для нее в форматной строке должен быть формат строкового поля.

Остальные символы, указанные в форматной строке, отличные от управляющих, при выводе изображаются так, как они представлены в этой строке, т. е. в том виде и в тех же позициях, в которых они указаны. Их используют в основном для вывода на печать заголовков и другой текстовой информации. Старые соглашения, принятые для оператора PRINT по использованию разделительной точки и точки с запятой, в операторе PRINT USING не действуют.

Числовые поля. При выводе числовой информации могут быть использованы форматные символы, образующие формат вывода числовых значений:

"##.##" — каждая цифра выводимого числа в формате представляется знаком #.

Десятичная точка отделяет целую часть от дробной. При этом в поле для вывода может быть указано любое расположение десятичной точки. Для вывода целых чисел десятичная точка в формате может отсутствовать.

Например по операторам

```
20 PRINT USING "##.###",75.634
30 PRINT USING "###",520
```

на терминал будет выведено

```
 75. 634
 520
```

Целая часть числа в отведенном поле всегда располагается справа, т. е. если количество цифр целой части числа меньше указанного в формате поля, то незанятые позиции занимаются пробелами. Если количество позиций, указанное для дробной части, недостаточно, то число округляется (не усекается). Если количество позиций для дробной части больше требуемого, то лишние позиции заполняются нулями.

Для отрицательных чисел необходимо предусматривать место под знак. Знак «—» всегда записывается слева от старшей значащей цифры; знак «+» при выводе положительных чисел не печатается. Например, операторы

```
50 PRINT USING "####.##",97.35
60 PRINT USING "##.##",56.746
70 PRINT USING "####.####",-25.34
```

выводят на терминал значения:

```
97.35
56.75
—25.34
```

Если выводимые данные не помещаются в заданное форматом поле, то для обозначения ошибки перед числом печатается знак «%», а затем — выводимое значение без соблюдения формата. Кроме того, если при округлении для числа потребуется бóльшее поле, чем отведено, то перед округленным числом также будет выведен знак «%». Например, при выполнении операторов

```
100 PRINT USING "##.##",126.94
110 PRINT USING "#.##",9.999999
```

будет отпечатано

```
% 126.94
% 10
```

В сомнительных случаях рекомендуется задавать поле несколько больших размеров, чем требуется. Порядок выводимых значений определяется порядком следования соответствующих элементов в списке.

Между элементами списка и форматами полей должно быть взаимно-однозначное соответствие. В случае, если в форматной строке указано больше форматов полей, чем элементов в списке, то оставшиеся форматы не используются. Если же в форматной строке указано меньше форматов полей, чем количество подлежащих выводу элементов, то применение форматов, содержащихся в форматной строке, не будет последовательно повторяться сначала до тех пор, пока список не будет полностью исчерпан. При этом каждое новое повторение форматной строки вызывает печать элементов списка с новой строки. Если в списке следует подряд несколько данных, имеющих числовые значения, то форматы полей в форматной строке нужно разделять знаками, отличными от знаков управления форматом либо знаками пробела. Это следует делать для того, чтобы несколько подряд идущих форматов в форматной строке не были понятны БЕЙСИК-системной как один формат. Например,

```
20 LET A=438.4
30 LET X=15.76\LET Y=120.496
40 LET X1=X-10\LET Y1=Y-100
50 PRINT USING "##.##",25.73,A
60 PRINT USING "X=##.##  Y=###.##",X,Y,X1,Y1
```

Сообщение, выводимое на терминал:

```
25.73
% 438
X=15.76  Y=120.50
X=5.76   Y=20.50
```

Форматы

```
"+##.##"
"##.##-"
"#,#.  ##"
```

Формат числового поля, кроме знаков # и «.», может содержать еще знаки «—» и «,». Если поле числа в форматной строке завершается знаком «—», то знак выводимого отрицательного числа печатается позади числа. Например, операторы

```
20 PRINT USING "###.#-",3.
30 PRINT USING "###.#-",-15.4
```

выведут на терминал значения

```
3.0
15.4 —
```

Запятая слева от десятичной точки означает, что целая часть числа будет разбита на группы по три цифры, начиная от десятичной точки, т. е. запятой будут определяться единицы от тысяч, тысячи от миллионов и т. д. Запятая справа от десятичной точки не является символом формата числового поля и ее присутствие в формате вывода рассматривается как обычный символ. Например, выполнение операторов

```
100 PRINT USING "####,###.#",1876234.14
110 PRINT USING   "###.#,#",35.485
120 PRINT USING  "####,#",25560
130 PRINT USING  "###,#.####",1234.5678
```

приведет к выдаче на терминал значений:

```
1,876,234.1
  35.5,
25,560
1,234.5678
```

Для вывода вещественных чисел в формате с десятичным порядком используется формат: "##.##^^^^"

Здесь четыре стрелки ^^^^ отводят место под букву E, знак показателя степени и две позиции — под показатель степени. Положение десятичной точки среди знаков # может быть любым. При этом

значащие цифры числа располагаются согласно части формата ##.## #, а показатель степени корректируется. Например,

```
30 PRINT USING "##.##^^^^",1875
40 PRINT USING "#.###^^^^",1875
50 RPINT USING "##.#^^^^",-0.256
60 LET F¤="#.##^^^^"
70 PRINT USING F¤,25.3
```

Вывод на терминал

```
18.75E+02
1.875E+03
—2.6E—01
2.53E+01
```

Строковые поля. В списке выводимых данных, кроме тех, которые имеют числовое значение, могут находиться и строковые данные. Для них в форматной строке используются специальные форматы, которые могут выводить необходимое количество символов, выравнивать выводимые данные по левому и правому краю поля, или располагать выводимые данные по центру отведенного строкового поля. Если символов в выводимой строке больше, чем резервируется позиций форматной строкой, выводимая строка усекается. Признаком начала строкового поля служит «'» — апостроф*. Например, операторы

```
40 PRINT USING "'","НЕТ"
50 PRINT USING "''","ABC","DEF"
```

выводят сообщение:

```
Н
AD
```

Для выравнивания строки по левой границе отведенного поля используется символ L в форматной строке, который следует за апострофом. Символ L также резервирует место для одного выводимого символа.

Таким образом, если для выводимой строки надо зарезервировать n символов и прижать эту строку к левой границе отведенного поля, то за символом апостроф следует расположить n символов L. Например, по операторам

```
20 PRINT USING "'L","А"
30 RPINT USING "'LLLL","ПЕТРОВ"
40 PRINT USING " 'LLL","ДА"
```

будет выведено

```
А
ПЕТР
ДА
```

---

* В версии БЕЙСИК-MSX для вывода первого символа строки служит «!» — восклицательный знак

Для выравнивания выводимой строки по правой границе отведенного поля используется символ R, который следует за апострофом. Так, согласно операторам

```
20 PRINT USING "'RRRRR", "A"
30 PRINT USING "'RRRRR" "ПЕТРОВ"
40 LET    F$=    "RRRRRR"
50 LET    C$=    "FI"
60 PRINT USING F$, C$
```

сообщение на терминале будет выглядеть следующим образом:

```
     A
ПЕТРОВ
    IF
```

Для расположения выводимой строки по центру отведенного поля применяется символ «С», который следует за апострофом. Правила его применения аналогичны правилам применения символов L и R в форматной строке. Например, операторы

```
40 PRINT USING "'CCCC","A"
50 PRINT USING "'CCCC","AB"
60 PRINT USING "'CCCC","ABC"
```

организуют вывод на терминал данных в следующем виде:

```
A
AB
ABC
```

Правильное сочетание строковых и числовых данных с соответствующими форматами полей дает возможность пользователю получить результаты вычислений в наиболее удобной для восприятия и документирования форме.

4.6. ЛОГИЧЕСКИЕ ОПЕРАЦИИ

В ряде версий языка БЕЙСИК допускается использование логических операций NOT, AND, OR, XOR, EQV, IMP. Логические операции выполняются над логическими операндами. Логические операнды могут принимать только два значения: «ИСТИНА» (или логическая 1) и «ЛОЖЬ» (или логический 0). Примером логического операнда может служить результат сравнения при выполнении операции отношения над числами или символами.

Обозначим логические операнды буквами A и B. Запишем в виде таблиц (табл. 4.1—4.6) результаты выполнения логических операций при различных комбинациях значений логических операндов.

Таблица 4.1. **Операция NOT (логическое Не, логическое отрицание)**

A	NOT A
0	1
1	0

Таблица 4.2. **Операция AND (логическое И, логическое умножение)**

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Таблица 4.3. **Операция OR (операция ИЛИ, логическое сложение)**

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 4.4. **Операция XOR (исключающее ИЛИ)**

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Таблица 4.5. **Операция EQV (логическая эквивалентность)**

A	B	A EQV B
0	0	1
0	1	0
1	0	0
1	1	1

Таблица 4.6. **Операция IMP (импликация)**

A	B	A IMP B
0	0	1
0	1	1
1	0	0
1	1	1

Данные таблицы называют *таблицами истинности* логических операций. Эти таблицы полностью определяют каждую из перечисленных логических операций.

Интересно отметить, что самое сложное логическое высказывание можно выразить с помощью трех основных логических операций NOT (НЕ), AND (И) и OR (ИЛИ).

Результат выполнения логических операций можно рассматривать как «ИСТИНУ» (ненулевой результат) или «ЛОЖЬ» (результат 0). Поэтому логические операции могут связывать несколько операций отношения в операторах условного перехода IF и выдавать значение «ИСТИНА» или «ЛОЖЬ» для принятия решений.

Примеры:

```
20 IF A=80 OR B<60 THEN 60
40 IF B¤<A¤ AND C¤="BAS" THEN PRINT B¤
50 IF NOT P THEN 120
```

В смешанных выражениях логические операции выполняются после арифметических операций и операций отношения. Очередность выполнения логических операций осуществляется согласно следующим приоритетам: 1. NOT; 2. AND; 3. OR; 4. XOR; 5. IMP; 6. EQV.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Какие операции можно выполнять со строковыми данными?
2. Как выполняется операция сравнения строк?
3. Какие стандартные функции предусмотрены в языке БЕЙСИК для обработки строк?
4. Как осуществляется ввод — вывод строковых данных?
5. Что такое нестандартные функции, определяемые пользователем?
6. Какова связь и в чем разница между формальными и фактическими аргументами?
7. Что такое подпрограмма?
8. Какие операторы используются для обращения к подпрограмме и для выхода из нее?
9. Какие операторы служат для подготовки к работе в графическом режиме?
10. Для чего предназначены операторы LINE, CIRCLE, PSET?

Глава 5

РАБОТА С ФАЙЛАМИ

5.1. ФОРМИРОВАНИЕ И ХРАНЕНИЕ ПРОГРАММ И ДАННЫХ

Рассмотрим на упрощенной схеме (рис. 5.1) формирование и хранение программы на языке БЕЙСИК. Когда пользователь садится за пульт микроЭВМ, он, как правило, не знает, какая программа находится в памяти, работала программа или нет, в каком состоянии остановлена ЭВМ и т. д. Состояние микроЭВМ перед началом работы изображено на рис. 5.1, *а*. Здесь *МП* — микропроцессор; *СМ* — системная магистраль для передачи данных, адресов и управляющей информации; *УВВ* — устройство ввода — вывода информации (терминал), посредством которого пользователь общается с микроЭВМ; *ОП* — оперативная память, в которой могут храниться программы и данные; *ВЗУ* — внешнее запоминающее устройство (накопитель на магнитных дисках), предназначенное для длительного хранения информации. На *УВВ* еще не набрано никакой информации, *ОП* заполнена информацией от предыдущих задач, либо находится в некотором неизвестном состоянии (после включения машины); *ВЗУ* свободно, т. е. там пока тоже не записано никакой информации.

Вычислительная машина способна воспринимать и выполнять не только операторы программы, но и некоторые другие инструкции, называемые *системными командами*. С тремя из них мы уже знакомы. Это команды RUN, NEW и LIST. Рассмотрим более подробно команду NEW (*новый*). Формат команды

NEW ⟨*имя программы*⟩

где ⟨*имя программы*⟩ — начинающаяся с буквы последовательность латинских букв и цифр, длина которой не более шести символов.

Пусть пользователь набирает на *УВВ* (рис. 5.1, *б*) эту команду в виде

NEW PROG1 ⟨ВК⟩

После нажатия клавиши ⟨ВК⟩ имя программы PROG1 заносится в оперативную память *ОП*, которая при этом очищается от предыдущей информации. После выполнения указанной команды *ОП* находится в распоряжении программы, которая имеет имя PROG1. Пользователю выдается сообщение READY, свидетельствующее о том, что ЭВМ готова принимать последующую информацию. Обратите внимание, что самой программы пока нет.

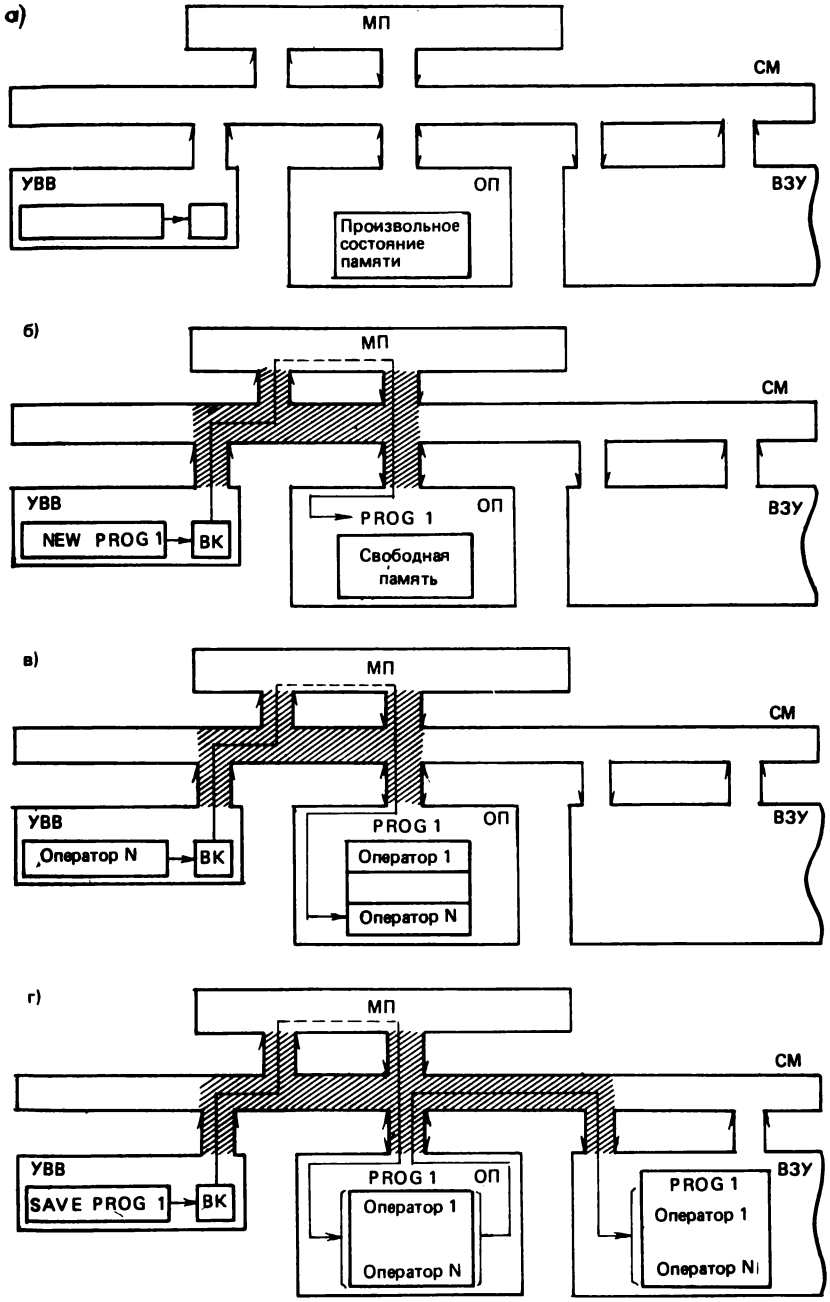


Рис. 5.1. Схема формирования и хранения программы на устройствах микроЭВМ

Далее пользователь начинает формировать свою программу. Он набирает на *УВВ* оператор *1* и по ⟨ВК⟩ помещает его в *ОП*, далее набирается и записывается оператор *2, 3,* ... и, наконец, последний оператор *N* помещается в *ОП* (рис. 5.1, *в*). Таким образом, вся программа сформирована и находится в *ОП*. После этого пользователь может выполнить программу, набрав на *УВВ* команду RUN.

Известно, что после выключения ЭВМ вся информация в *ОП*, а следовательно, и наша программа PROG1 пропадают. Кроме того, очередной пользователь, набрав команду NEW, так же может уничтожить нашу программу. Чтобы сохранить программу для последующего использования или корректировок, в языке БЕЙСИК имеется специальная системная команда SAVE (*сохранить* или, буквально, *спасти*). Формат команды:

SAVE ⟨*имя программы*⟩

Здесь ⟨*имя программы*⟩ такое же, как в команде NEW. Если набрать на *УВВ* (рис. 5.1, *г*) эту команду в виде

SAVE PROG1 ⟨ВК⟩

то после нажатия клавиши ⟨ВК⟩ она выполнится, и все операторы программы PROG1 из *ОП* перепишутся на магнитный диск в *ВЗУ*. Кроме того, программа PROG1 останется и в *ОП*.

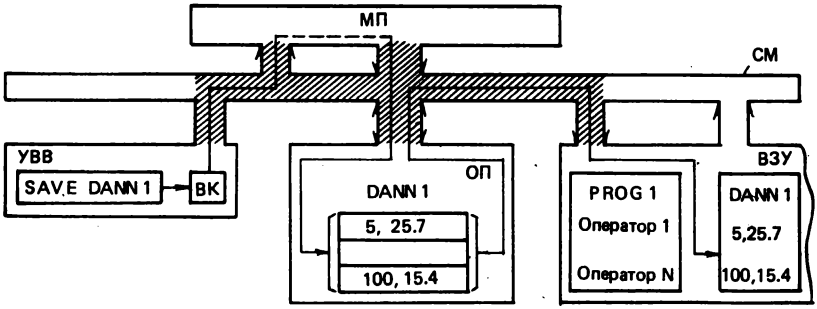


Рис. 5.2. Схема сохранения набора данных на ВЗУ

Ранее было отмечено, что по способу хранения ЭВМ не делает различия между командами и данными. Вместо операторов программы на *УВВ* можно набирать некоторые числа и после нажатия ⟨ВК⟩ вводить их в *ОП*. Естественно, что такой числовой массив следует как-то озаглавить, например DANN1, и вводить значения после команды NEW DANN1 аналогично вводу программы (рис. 5.1, *б, в*). При вводе массива данных с терминала следует соблюдать правило, чтобы каждая строка массива данных начиналась с целого числа, подобно номеру оператора. Этот массив также можно сохранить на магнитном диске с помощью команды SAVE DANN1 (рис. 5.2). При этом программа PROG1, записанная ранее, также сохраняется. Таким образом, на *ВЗУ* можно хранить достаточно большое число наборов данных и программ.

5.2. ПОНЯТИЕ ФАЙЛА

Совокупность данных, объединенных по некоторому смысловому признаку, называется *файлом*. Файлами являются как набор операторов языка БЕЙСИК, так и наборы данных (числовых или строковых), которые могут использоваться при решении задачи. В этом смысле упомянутые выше программа PROG1 и массив DANN1 являются файлами. Как правило, они хранятся на внешних носителях информации (магнитных дисках и лентах).

Файл, приведенный ниже, состоит из пяти операторов языка БЕЙСИК:

```
10 LET A=2
20 LET B=3
30 LET C=A+B/2
40 PRINT A,B,C
50 END
```

Он содержит команды для ЭВМ и называется *программным файлом*. Рассмотрим другой файл, представленный таблицей:

1,	4,	5,	5,	5
2,	3,	3,	4,	3
3,	2,	2,	3,	3
4,	5,	5,	5,	5

Он содержит обычные числа, которые можно, например, толковать как порядковый номер учащегося и его текущие отметки. Файл, содержащий числа или другие данные, с которыми работает программа, называется *файлом данных*. Как программный файл, так и файл данных могут содержать столько строк, сколько нужно пользователю: от одной или двух строк до сотен и даже тысяч. Отдельная величина, входящая в файл, называется *элементом* или *компонентой файла*. Именем файла может быть последовательность букв и цифр (от 1 до 6), начинающаяся с буквы. В ОП можно формировать и хранить только один файл, в ВЗУ можно хранить много файлов.

Вычисления по программе, ее изменение и другие действия с содержимым файлов можно выполнять только тогда, когда соответствующий файл (программный или файл данных) находится в ОЗУ.

5.3. ОБРАБОТКА ФАЙЛОВ

Ранее было показано, что данные в программу можно ввести различными способами:

1) встроить данные в программу, например 20 LET A = 5;

2) прочитать данные оператором READ из блока данных, определяемого оператором DATA;

3) ввести данные с терминала с помощью оператора INPUT.

Кроме указанных способов программа может получать данные из файла данных, находящегося на ВЗУ.

Передать в программу информацию, содержащуюся в файле данных, — это значит присвоить значения элементов файла переменным или массивам программы.

Имена файлов, под которыми они хранятся на ВЗУ, называются *системными именами файлов*. В программе на языке БЕЙСИК их нельзя применять. В языке БЕЙСИК имеются специальные имена для файлов:

$$\#1, \#2, \ldots, \#12$$

где # — признак программного файла; 1, 2, ..., 12 — указатель файла.

Перед обращением из программы к файлу данных необходимо установить соответствие между файлом на ВЗУ и файлом в программе. Это значит установить связь между набором данных, который записан на внешнем посителе и имеет некоторое имя (например, DANN1) с набором данных, которым будет оперировать программа и который также имеет некоторое имя (например, #1). Эта связь в программе устанавливается оператором открытия файла OPEN. Например, по оператору

```
20 OPEN "DANN1" FOR INPUT AS FILE #1
```

устанавливается связь между файлом DANN1 и файлом #1 для чтения файла DANN1 из ВЗУ в программу. Подробнее оператор OPEN будет рассмотрен ниже. Обращение для чтения к файлу #1 автоматически вызовет обращение к данным файла на ВЗУ с именем DANN1. Таким образом, к файлу DANN1 в программе можно обращаться, пользуясь именем #1.

Основными этапами при работе с файлами являются:

установление связи между файлом в программе и файлом на ВЗУ;

чтение данных из файла на ВЗУ в переменные и массивы программы, используя понятные программе имена файлов (#1, #2, ..., #12);

работа с полученными данными;

запись выходных результатов из переменных и массивов программы с помощью понятных программе имен файлов в файлы на внешних носителях в случае необходимости длительного хранения;

прекращение связи между файлом в программе и файлом на ВЗУ.

Эти этапы выполняются с помощью операторов работы с файлами.

Оператор открытия файлов OPEN. Перед обращением к файлу его необходимо открыть, т. е. связать файл на ВЗУ с файлом в программе. Формат оператора

OPEN ⟨*сист. имя*⟩ FOR INPUT AS FILE ⟨*прогр. имя*⟩

или

OPEN ⟨*сист. имя*⟩ FOR OUTPUT AS FILE ⟨*прогр. имя*⟩

где OPEN — имя оператора (*открыть*); ⟨*сист. имя*⟩ — имя файла на ВЗУ (последовательность от 1 до 6 символов, начинающаяся с буквы

и заключенная в кавычки); FOR INPUT — служебные слова, указывающие функцию файла (*для ввода*); AS FILE — служебные слова (*как файл*); ⟨*прогр. имя*⟩ — имя файла в программе (#1, # 2, ..., #12) # — признак файла, используемого в программе; 1, 2, ..., 12 — указатель файла (указателем файла вместо константы может служить и арифметическое выражение, значение которого находится в диапазоне от 1 до 12); БЕЙСИК-система разрешает одному пользователю держать открытыми не более 12 файлов; FOR OUTPUT — служебные слова, указывающие функцию файла (*для вывода*).

Оператор OPEN со словами FOR INPUT открывает существующий файл, хранящийся на некотором ВЗУ, для считывания информации из него в ОП.

Оператор OPEN со словами FOR OUTPUT создает на ВЗУ новый файл с именем ⟨*сист. имя*⟩. Если на устройстве существует файл с тем же именем, он уничтожается. Например,

```
40 OPEN "DANN" FOR INPUT AS FILE #1
250 OPEN "REZULT"FOR OUTPUT AS FILE #2
```

Одна ЭВМ может иметь несколько ВЗУ, в том числе и однотипных. Чтобы точно указать, на каком именно устройстве находится нужный файл, формат системного имени может быть расширенным, т. е. содержать дополнительную информацию:

⟨*устройство*⟩ : ⟨*сист. имя*⟩

где ⟨*устройство*⟩ — имя устройства, на котором находится или будет создаваться файл.

Например, для ДВК

DX1 : DANN1	Файл DANN1 находится на устройстве номер 1 с гибким магнитным диском (номера устройств могут быть от 0 до 3)
LP : DANN2	Файл DANN2 будет выводиться на устройство печати

Пример оператора OPEN, использующего расширенное имя системного файла и указатель файла в виде переменной:

```
300 OPEN "LP:DANN2" FOR OUTPUTAS FILE #L
```

Расширения имени могут несколько отличаться для разных ЭВМ.

Оператор закрытия файлов CLOSE. После завершения обращений к файлам их необходимо закрыть. Для закрытия файлов используется оператор CLOSE. Формат оператора

CLOSE ⟨*прогр. имя 1*⟩, ..., ⟨*прогр. имя* N⟩

где CLOSE — имя оператора (*закрыть*); ⟨*прогр. имя 1*⟩, ..., ⟨*прогр. имя* N⟩ — имена файлов в программе.

Указанное в операторе CLOSE имя должно быть таким же, как и имя в операторе OPEN после слов AS FILE. оно означает имя закрываемого файла. С помощью одного оператора CLOSE можно закрыть любое количество файлов. Если закрывается несколько файлов,

то их имена в этом операторе разделяются запятыми. Например, оператор 120 во фрагменте

```
110 LET N=10
120 CLOSE #1,#3,#N
```

закрывает файлы с именами #1, #3, #N.

Оператор CLOSE без списка имен файлов закрывает все открытые файлы. После выполнения данного оператора связь между файлом на ВЗУ и файлом в программе разрывается.

Операторы обмена данными.

При работе с файлами следует учитывать, что:

1) чтение из файла не разрушает данные. Файл на внешнем запоминающем устройстве сохраняется;

2) запись в файл с именем, которого нет на ВЗУ, создает на ВЗУ новый файл;

3) запись в файл с именем, которое есть на ВЗУ, уничтожает старый файл с этим именем и организует новый с таким же именем;

4) несколько программ последовательно могут обращаться к одному и тому же файлу на ВЗУ. Внутри каждой программы этот файл может обозначаться собственным именем (#1, #2, ..., #12) в соответствии с оператором OPEN.

Оператор INPUT#. Для считывания данных в оперативную память из файла на ВЗУ используется оператор INPUT#. Операции считывания данных из файла допускаются лишь после его открытия. Формат оператора

INPUT ⟨*прогр. имя*⟩, v1, v2, ..., vN,

где INPUT — имя оператора (*ввести*); ⟨*прогр. имя*⟩ — имя файла в программе (#1, #2, .., #12); v1, v2, ..., vN — список имен переменных программы, которым присваиваются вводимые значения из файла на ВЗУ. Имена переменных отделяются друг от друга запятыми. Например,

```
30  OPEN "DAN5" FOR INPUT AS FILE #1
40  INPUT #1 , A,B,C
```

Здесь оператор 30 связывает файл DAN5 на ВЗУ с файлом #1 в программе, затем оператором 40 считываются числовые значения из файла с именем #1 в переменные A, B, C. Иными словами, числовые значения из файла DAN5 присваиваются переменным A, B, C. Элементы в файле на ВЗУ должны быть разделены запятой или ⟨ВК⟩.

Для считывания из ВЗУ файлов, содержащих строковые данные, в некоторых версиях языка используется оператор

LINPUT#,

формат которого аналогичен формату оператора INPUT#. В этом случае v1, v2, ..., vN — имена строковых переменных. В файле на ВЗУ строковые переменные должны отделяться друг от друга ⟨ВК⟩.

Оператор PRINT #. Как было показано ранее, файл данных можно сформировать с терминала на ВЗУ с помощью команд NEW и SAVE. Если в программе получена некоторая совокупность данных, то из них можно сформировать файл на ВЗУ с помощью оператора PRINT#. Формат оператора:

PRINT ⟨*прогр. имя*⟩, e1, e2, ..., eN,

где PRINT — имя оператора; ⟨*прогр. имя*⟩ — имя файла в программе (#1, #2, ..., #12), ранее открытого для вывода; e1, e2, ..., eN — список констант, переменных либо арифметических выражений, значения которых пересылаются в файл на ВЗУ.

Этот оператор записывает значения величин, указанных в списке оператора, с помощью программного файла с именем ⟨*прогр. имя*⟩ в файл на ВЗУ с именем ⟨*сист. имя*⟩. Предварительно нужно установить связь между файлами с именами ⟨*прогр. имя*⟩ и ⟨*сост. имя*⟩ на некотором устройстве с помощью оператора OPEN FOR OUTPUT. Если таким устройством названо, например LP:, то оператор PRINT# выводит данные на автоматическое цифровое печатающее устройство; если MX1:, то файл формируется на устройстве с малым гибким магнитным диском с номером 1; если DX0, то на устройстве с большим гибким магнитным диском с номером 0. Например,

```
200 OPEN "DX3:DANN# FOR OUTPUT AS FILE #2
210 PRINT #2 , A,B,5
```

Здесь оператором 210 числовые значения переменных A, B и константа 5 будут записаны в файл DANN на устройство с гибким магнитным диском номер 3 с помощью программного файла #2.

В списке оператора PRINT # могут употребляться символьные константы, переменные или выражения, например

230 PRINT#5, "НАЧАЛО ФАЙЛА"

Употребление операторов PRINT# и INPUT # должно удовлетворять следующим правилам:

1. Оператор INPUT#, считывающий данные с некоторого файла ВЗУ, должен дублировать формат оператора PRINT#, записавшего данные в этот файл, т. е. переменные в списках этих операторов должны согласовываться по количеству и типу.

2. Значения величин при считывании данных из файлов ВЗУ с помощью оператора INPUT # должны быть отделены друг от друга запятой или с помощью ⟨ВК⟩. В противном случае эти данные будут восприниматься оператором INPUT# как одно значение. Поэтому при создании файлов, содержимое которых предполагается использовать в этой или другой программе, необходимо ставить запятые для разделения данных при их записи в файл оператором PRINT#. Таким образом, оператор PRINT# должен содержать символьные константы «", "» между элементами списка. Элементы списка оператора PRINT#, включая символьную константу «", "», разделяются запятыми.

Рассмотрим пример работы с файлами:

```
30 LET A=53.4 \ B¤="КНИГА" \ C%=25
40 OPEN "F1" FOR OUTPUT AS FILE #2
50 PRINT #2,A,",",B¤,",",C%
60 CLOSE #2
70 OPEN "MX1:F1"  FOR INPUT AS FILE #4
80 INPUT #4,A1,B1¤,C1%
90 PRINT "A1="; A1, "B1¤="; B1, "C1%="; C1%
100 END
RUN


A1= 53.4        B1¤=КНИГА       C1%= 25


READY
```

Здесь оператор 50 создает файл вида:

53.4, КНИГА, 25

Оператор 80 считывает эти три значения из файла соответственно в переменные A1, B1 ¤, C1%, оператор 90 значения этих переменных для проверки выводит на печать. Операторы 50 и 80 содержат одинаковое количество переменных (три), типы которых взаимно согласованы: A и A1 — вещественные, B ¤ и B1 ¤ — символьные, C% и C1% — целые.

Оператор IF END #. Он передает управление при обнаружении признака конца файла.

Формат оператора:

IF END ⟨*прогр. имя*⟩ THEN m1

либо

IF END ⟨*прогр. имя*⟩ THEN ⟨*оператор*⟩

где IF END — имя оператора (*если конец файла*); ⟨*прогр. имя*⟩ — имя файла в программе (. 1, # 2, ..., # 12); m1 — номер оператора, которому передается управление при обнаружении конца файла; ⟨*оператор*⟩ — оператор, выполняемый при обнаружении конца файла.

Например,

120 IF END # 2 THEN 250

Здесь при обнаружении конца файла #2 управление будет передано оператору 250. В противном случае будет выполнен оператор, следующий за IF. Признак конца файла обнаруживается, если в файле больше нет данных.

Оператор IF END # применим только к файлам последовательного доступа. Этим оператором удобно пользоваться, когда количество элементов файла заранее неизвестно.

5.4. ФАЙЛЫ ПОСЛЕДОВАТЕЛЬНОГО И ПРЯМОГО ДОСТУПОВ

Файл последовательного доступа — это такой файл, из которого данные можно получать лишь строго в такой последовательности, в какой они в нем расположены. Например, если файл последовательного доступа содержит числа 2, 4, 10, 15, 7 в указанном порядке, то считать из него число 4 можно, лишь предварительно считав число 2, а чтобы считать число 15, необходимо считать числа 2, 4 и 10. Все рассмотренные выше файлы и работа с ними относились к файлам последовательного доступа.

Файл прямого доступа к данным, хранящимся на диске, — это такой файл, который позволяет считывать из него данные в любом порядке. С этой целью каждый элемент файла имеет номер, как индекс в массиве. Считывание (запись) данных происходит в соответствии с этим номером. Чтобы определить файл как файл прямого доступа, его необходимо описать в операторе DIM#. Формат оператора:

DIM ⟨*прогр. имя*⟩, ⟨*имя массива*⟩

где DIM — имя оператора, ⟨*прогр. имя*⟩ — имя файла в программе (#1, #2, ..., #12); ⟨*имя массива*⟩ — имя массива, максимальные размеры и организация которого отражают структуру файла.

С элементами файла можно работать как с элементами этого массива. Например,

40 DIM#1, A(25)

Этот оператор означает, что программный файл с именем #1 организован как одномерный массив A (25). Минимальный индекс массива равен нулю.

В операторе DIM # может быть указано не одно имя массива, а список имен. Например, оператор

50 DIM#2, B (50), C% (10, 10)

указывает, что файл # 2 организован следующим образом. В нем сначала располагаются 51 элемент вещественных чисел, организованных как одномерный массив B (50), а затем 121 элемент целых чисел, организованных как массив A (10, 10).

Перед употреблением оператора DIM # соответствующий файл на ВЗУ должен быть предварительно открыт с помощью оператора OPEN. Примеры работы с файлами прямого доступа:

```
10 OPEN "DAN2" FOR OUTPUT AS FILE #2
20 DIM #2,A(10)
30 FOR I=0 TO 10
40 A(I)=I
50 NEXT I
60 CLOSE #2
70 END
```

```
RUN

READY

 10 OPEN "DAN2" FOR INPUT AS FILE #2
 20 DIM #2,A(10)
 30 LET S=0
 40 FOR I=0 TO 10 STEP 2
 50 LET S=S+A(I)
 60 PRINT A(I)
 70 NEXT I
 80 PRINT "S=";S
 90 CLOSE #2
 100 END
 RUN


  0
  2
  4
  6
  8
  10
 S=30


 READY
```

Здесь первая программа создает на магнитном диске файл DAN2, а вторая программа выводит на экран значения четных элементов этого файла и их сумму S.

Данные в файле прямого доступа хранятся в особой форме, поэтому такие файлы нельзя формировать с терминала с помощью команд NEW—SAVE. Их нельзя также прочитать в ОЗУ командой OLD (см. ниже команду OLD).

5.5. РАБОТА С ФАЙЛАМИ

Рассмотрим работу с файлами на конкретном примере.

Пример 5.1. Имеются данные о контрольной работе учащихся в следующем виде: первое число обозначает номер учащегося в журнале группы, второе число — оценку за контрольную работу. Данные о контрольной работе записаны в файл с именем KRAB.

Найти средний балл группы по контрольной работе и указать количество учащихся в группе. Средний балл сохранить в файле SRBALL, а программу — в файле OBRAB для дальнейшего использования.

Этапы решения данной задачи.

1. Сформируем на терминале файл данных KRAB и запишем его на устройство с гибким магнитным диском:

```
NEW KRAB
1, 4 ⟨ВК⟩
2, 5 ⟨ВК⟩
3, 2 ⟨ВК⟩
4, 4 ⟨ВК⟩
5, 3 ⟨ВК⟩
SAVE KRAB ⟨ВК⟩
READY
```

Обратите внимание, что запятые, разделяющие числа и ⟨ВК⟩, записаны в файл KRAB уже при формировании файла в отличие от оператора PRINT, где запятая и ⟨ВК⟩ сами выступают в качестве записываемых символов.

2. Составим программу обработки, т. е. программу нахождения среднего балла; сформируем ее в ОЗУ для последующего исполнения. Известно, что средний балл равен сумме баллов, разделенных на количество учащихся.

В программе приняты следующие обозначения: S — переменная, в которой будет накапливаться сумма баллов; N — переменная, в которой хранится номер учащегося в журнале группы; B — переменная, в которой хранится оценка учащегося; I — переменная, в которой накапливается количество учащихся, оценки которых обработаны; C — переменная, в которой получается средний балл.

Текст программы и результаты ее работы приведены ниже.

```
10 REM ПРОГРАММА ОБРАБОТКИ ОЦЕНОК
20 LET C=0
30 LET I=0
40 OPEN "MX1:KRAB" FOR INPUT AS FILE #1
50 REM СЧИТЫВАНИЕ ДАННЫХ ИЗ ФАЙЛА KRAB
60 INPUT #1,N,B
70 LET I=I+1
80 LET S=S+B
90 IF END #1 THEN 110
100 GO TO 60
110 LET C=S/I
120 PRINT "СР.БАЛЛ ГР.=";C,"КОЛИЧ. УЧ.-";I
130 CLOSE #1
140 OPEN "MX1:SRBALL" FOR OUTPUT AS FILE #2
150 REM ЗАПИСЬ СР.БАЛЛА В ФАЙЛ НА ДИСКЕ
160 PRINT #2,"СРЕДНИЙ БАЛЛ",",",C
170 CLOSE #2
180 END
RUN

СР.БАЛЛ ГР.= 3.6            КОЛИЧ.УЧ.- 5


READY
```

3. Сохраним программный файл

```
SAVE OBRAB ⟨ВК⟩
READY
```

При выполнении последовательно пунктов 1, 2, 3 проводится обработка данных о контрольной работе и три файла: KRAB, SRBALL, OBRAB записываются на гибком магнитном диске для последующего использования.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Что такое файл?
2. В чем сходство и различие между программным файлом и файлом данных?
3. Какой оператор устанавливает соответствие между файлом на внешнем запоминающем устройстве и файлом в программе?
4. Каковы основные этапы работы с файлами?
5. Как считывать файл из ВЗУ в оперативную память?
6. Каково назначение оператора PRINT#?
7. В чем разница между файлом последовательного доступа и файлом прямого доступа?

Глава 6

СИСТЕМНЫЕ КОМАНДЫ И СРЕДСТВА ОТЛАДКИ ПРОГРАММ

6.1. ЯЗЫК СИСТЕМНЫХ КОМАНД

Наряду с языком программирования для общения с БЕЙСИК-системой пользователю необходимо знать определенный набор системных команд, используемых для управления работой системы. Как указывалось ранее, БЕЙСИК-система работает в двух режимах: программном и командном. Иногда их называют соответственно косвенным и непосредственным. В программном режиме выполняются операторы, имеющие номера (метки). Их выполнение начинается после ввода системной команды RUN (*пуск*).

В командный режим система переходит после ее запуска, после завершения выполнения программы, при обнаружении ошибки, при вводе специальной команды программистом. При переходе в этот режим на дисплее появляется сообщение READY (*готов*).

Командный режим обеспечивает следующие возможности:

ввод и выполнение системных команд и отдельных операторов языка БЕЙСИК, записанных без меток;

ввод программы и запоминание ее в памяти ЭВМ;

распечатку программы, находящейся в ЭВМ, на дисплее или устройстве печати;

сохранение программы на внешнем носителе информации (магнитном диске, магнитной ленте, перфоленте);

внесение изменений в программу (редактирование);

удаление программы (стирание) с внешнего носителя;

запуск программы на выполнение;

окончание работы пользователя с системой.

Кроме сообщений о состоянии системы она может выдавать пользователю запросы, на которые он обязан отвечать.

Рассмотрим основные системные команды. Характерным их признаком является отсутствие номера строки. В разных БЕЙСИК-системах такие команды могут несколько отличаться по написанию и содержанию от рассмотренных ниже и даже отсутствовать.

Команда NEW (*новая*) используется для создания новой программы. Она очищает содержимое памяти и присваивает имя программе, которая будет вводиться с дисплея в оперативную память ЭВМ. Данная команда была рассмотрена ранее. Если ввести команду NEW без имени программы, то на экране появляется сообщение:

NEW FILE NAME—

(*имя нового файла*). После этого программист должен указать имя программы или нажать клавишу ⟨ВК⟩. В последнем случае именем программы будет NO NAME (*без имени*).

С помощью команды NEW можно сформировать не только программу, но и файл данных (см. пример 5.1). При его формировании каждая строка файла должна обязательно начинаться с целого числа. В противном случае БЕЙСИК-система будет выдавать сообщение об ошибке.

Команда SCR (*ликвидировать*) — очищает содержимое оперативной памяти и присваивает имя NO NAME программе, вводимой с терминала.

Команда CLEAR (*очистить*) обнуляет содержимое переменных и массивов, описанных в программе. Имя программы не изменяется.

Команда RENAME (*переименовать*) изменяет имя программы, находящейся в оперативной памяти ЭВМ, на новое. Формат команды

RENAME ⟨*имя нов*⟩

где ⟨*имя нов*⟩ — новое имя программы.

Пример использования команд NEW и RENAME:

```
NEW PRIMER
10 INPUT X
20 PRINT SQR(X)
30 END
RENAME PRIM1
```

После ввода команды RENAME программа, имевшая имя PRIMER, будет называться PRIM1.

Команда DEL (*исключить*)* удаляет из программы операторы с указанными номерами. Она может иметь следующие формы.

DEL m1—m2	Исключить операторы, имеющие номера от m1 до m2 включительно
DEL m1—	Исключить все операторы, начиная с оператора, имеющего номер m1, и до конца программы
DEL—m2	Исключить все операторы от начала программы до оператора с номером m2 включительно
DEL	Удалить всю программу

Например, команда DEL 20—90 исключает из программы операторы с номерами 20, 90 и все операторы, номера которых находятся в интервале между ними. Для удаления одного оператора достаточно после DEL указать номер этого оператора, например DEL 40.

Команда SUB (*изменить подстроку*) корректирует часть строки. Формат команды:

SUB m ✱ *стр* 1 ✱ *стр* 2 ✱ N

где SUB — имя системной команды; m — номер редактируемой строки; ✱ — символ-ограничитель; *стр* 1 — последовательность заменяемых

* В БЕЙСИК-системе MSX с этой целью используется команда DELETE.

символов; *стр* 2 — последовательность символов, которые вновь вводятся в строку символов; N — число, указывающее, какой по счету символ подлежит замене, если в строке имеется несколько одинаковых символов (этот параметр можно не указывать, если в строке не имеется одинаковых символов, которые заменяются).

Ограничитель может быть любым символом, за исключением пробела, табуляции и цифр. Ограничитель не должен встречаться в *стр* 1 и *стр* 2. Например,

```
20 LET A=45.6          Редактируемая строка
SUB 20*4*36
20 LET A=365.6         Отредактированная строка
```

или

```
40 PRINT "ПЕРЕМЕТР ТРЕУГОЛЬНИКА"    Редактируемая строка
SUB 40*Е*И*2
40 PRINT "ПЕРИМЕТР ТРЕУГОЛЬНИКА     Отредактированная строка
```

Здесь в строке 40 вторая буква Е заменена на букву И.

Команда RESEQ (*перенумеровать*) перенумеровывает строки программы, находящейся в памяти. Это полезно делать, когда из программы удалялись или в нее вставлялись какие-то операторы. Формат команды

RESEQ m 1, m 2 — m 3, e

где m 1 — новый номер строки; m 2 — m 3 — номера строк от m 2 до m 3, которые перенумеровываются; e — шаг приращения новых номеров строк.

Приведенная ниже программа

```
120 LET A=3
140 LET B=SIN(A^2)
150 PRINT "B="; B
160 END
RESEQ 10  120-160, 10
```

по команде RESEQ преобразуется к виду:

```
10 LET A=3
20 LET B=SIN(A^2)
30 PRINT "B="; B
40 END
```

Команда LIST (*вывести текст программы*) выводит на экран дисплея программу, находящуюся в памяти ЭВМ. Эта команда имеет несколько форм:

LIST	Вывод всей программы
LIST m1—m2	Вывод строк, номера которых попадают в указанный диапазон
LIST m1—	Печать строк программы, начиная с номера m1
LIST—m2	Печать строк от начала программы до строки с номером m2 включительно
LIST m	Распечатка указанной строки

Например, команда

LIST 10—80

выводит на экран дисплея строки, имеющие номера от 10 до 80 и имя программы.

Команда LLIST выводит текст программы на печатающее устройство.

Команда RUN (*пуск*) служит для запуска на выполнение программы, находящейся в оперативной памяти. Этой командой мы уже пользовались. При ее выполнении на печать выводится заглавие программы, содержащее имя выполняемой программы, дату и время. Если эта информация не требуется, то можно воспользоваться командой RUNNH, действие которой аналогично RUN, но при выполнении ее не выводится заглавие программы.

Командой RUN можно также вызвать выполнение программы, находящейся в виде программного файла на внешнем запоминающем устройстве. В этом случае за словом RUN должно следовать имя файла, содержащего нужную программу.

Формат команды

RUN ⟨*имя программы*⟩

где ⟨*имя программы*⟩ — имя файла на внешнем запоминающем устройстве, в котором находится программа.

По этой команде система осуществляет поиск файла с именем ⟨*имя программы*⟩ на диске, и если обнаруживает его, то переписывает программу в оперативную память и выполняет ее. Например, по команде

RUN OBRAB

программа, хранящаяся на гибком магнитном диске под именем OBRAB, будет переписана в оперативную память и выполнена.

Команда SAVE (*сохранить*) записывает программу или данные, находящиеся в оперативной памяти ЭВМ, на ВЗУ с гибким магнитным диском. Данная команда рассмотрена в гл. 5. В случае наличия у ЭВМ нескольких однотипных запоминающих устройств с гибкими магнитными дисками либо необходимости сохранения программы на устройстве, отличном от устройства с гибким магнитным диском, имя программы в команде SAVE должно быть расширено:

⟨*устройство*⟩ : ⟨*имя программы*⟩

Например, SAVE DX3 : PROG1 либо SAVE CT : PROG.

По первой команде программа с именем PROG1 будет сохранена на устройстве с гибким магнитным диском номер три, а по второй — на кассетном магнитофоне. Для вывода текста программы на печатающее устройство (имя устройства LP) используется команда

SAVE LP : PROG1

Перед тем как записать файл на ВЗУ по команде SAVE, БЕЙСИК-система проверяет, нет ли на ВЗУ файла с таким же именем. В слу-

чае его обнаружения она выдает сообщение, предлагающее пользователю переименовать записываемый файл. Это делается для того, чтобы случайно не стереть нужный файл.

Если же все-таки необходимо записать на ВЗУ файл под тем же именем, то его предварительно удаляют с помощью команды UNSAVE. Это бывает необходимо при многократных корректировках одной и той же программы.

Команда UNSAVE (*удалить*) служит для удаления файла с внешнего запоминающего устройства. Формат команды:

UNSAVE ⟨*устройство*⟩ : ⟨*имя файла*⟩

Если файл удаляется с гибкого магнитного диска, то имя устройства не нужно указывать. Например, UNSAVE OBRAB удаляет файл с именем OBRAB.

Команда REPLACE (*заменить*) аналогична команде SAVE, но команда REPLACE заменяет файл, ранее созданный командой SAVE. Формат команды:

REPLACE ⟨*имя*⟩

где ⟨*имя*⟩ — имя файла, который замещает хранящийся файл на ВЗУ с таким же именем.

При использовании команды REPLACE, в отличие от SAVE специально удалять заменяемый файл с ВЗУ не требуется. В случае отсутствия на ВЗУ файла, который замещается, возникает ошибка и выдается сообщение об ошибке.

Команда OLD (*вызвать старую программу*)* служит для помещения в оперативную память ЭВМ программы или данных, хранящихся на гибком магнитном диске. Формат команды

OLD ⟨*имя*⟩

где ⟨*имя*⟩ — системное имя программы или файла данных, хранящихся на гибком магнитном диске.

Например, команда

OLD OBRAB

вызывает и помещает в оперативную память программу, записанную ранее на гибкий магнитный диск под именем OBRAB. Далее эту программу можно исполнять либо редактировать по усмотрению пользователя. Часто эта команда используется после выполнения команды SAVE (*сохранить*). Следует учитывать, что файл данных, сформированный в программе командой PRINT # и записанный на ВЗУ под некоторым именем, также может быть считан в ОЗУ командой OLD ⟨*имя*⟩. Это возможно, если каждая строка записанного файла начинается с целого числа. В противном случае система выдает сообщение об ошибке. Считать файл последовательного доступа, начинающийся не с целого числа, можно в программе оператором INPUT#.

Команда BYE служит для прекращения работы с БЕЙСИК-системой.

---

* В БЕЙСИК-системе MSX с этой целью служит команда LOAD ⟨*имя*⟩.

6.2. КОМАНДЫ КЛАВИАТУРЫ

Управление работой системы может осуществляться с клавиатуры терминала. Ряд команд можно подать, нажав на определенные управляющие клавиши. Так, с помощью клавиши ВК (в некоторых системах это — CR, ENTER, RETURN) заканчивается набранная строка и вводится в операционную память ЭВМ. Если программа выполняется очень долго, то ее можно остановить командой СУ/С (CTRL/C). Для ввода команды нужно одновременно нажать клавиши СУ (CTRL) и С. Команда СУ/С, введенная дважды, немедленно останавливает выполнение программы. На экран выдается сообщение:

STOP AT LINE XXXX

где XXXX — номер оператора, выполнение которого было прервано.

Приостановить вывод текста программы или результатов выполнения программы на экран дисплея можно командой СУ/S, одновременно нажав клавиши СУ и S. Это дает возможность пользователю проанализировать полученные данные перед тем, как их заменят новые. Для продолжения вывода используют команду СУ/Q, для чего следует одновременно нажать клавиши СУ и Q. Перечень основных системных команд приведен в приложении 2.

6.3. ОТЛАДКА ПРОГРАММ В РЕЖИМЕ ДИАЛОГА

Взаимодействие человека и ЭВМ может происходить в форме диалога. Например, после набора программы на терминале и подачи команды RUN программа может быть либо выполнена и пользователю будут выданы машиной результаты счета, либо будет выдано сообщение об ошибке. В случае верных результатов пользователь может перейти к другой работе на ЭВМ, задав ей соответствующую команду. В случае ошибки следует исправить ошибочные операторы и дать указание машине, чтобы она повторно выполнила исправленную программу. Это и есть простейший диалог человека и ЭВМ. Готовность ЭВМ принимать информацию от пользователя сигнализируется либо сообщением READY (*готов*), либо знаком «?», либо текстом вопроса.

Программы можно корректировать следующим образом. Для исправления оператора нужно набрать на терминале правильный оператор под тем же номером. Можно также воспользоваться командой SUB.

Для удаления оператора следует набрать его номер и нажать клавишу возврата каретки ⟨ВК⟩.

Для вставления дополнительного оператора между операторами с

номерами m1 и m2 нужно набрать вставляемый оператор с номером, находящимся в интервале от m1 до m2.

Удаление набираемого символа производится нажатием клавиши DEL (*забой*).

Как указывалось ранее, БЕЙСИК-система обеспечивает возможность непосредственного выполнения некоторых операторов сразу после их введения в оперативную память. Для указания системе о работе в таком режиме служит введение оператора без номера строки. По окончании выполнения оператора в непосредственном режиме система выдает сообщение READY. Следует учитывать, что, как правило, в непосредственном режиме одновременно может быть введен и выполнен только один исполняемый оператор языка БЕЙСИК. В некоторых системах (например, ДВК) разрешается запись нескольких операторов непосредственного режима в одной строке. В этом случае операторы разделяются символом «\». Например,

```
LET A = 3\LET B = 5\PRINT A + B, A * B ⟨ВК⟩
8             15
READY
```

Выполнение операторов в непосредственном режиме особенно полезно при выполнении простых вычислений и отладке программ. В непосредственном режиме можно узнать значение любой интересующей переменной, использовав оператор PRINT. Можно присвоить любое значение переменной или изменить его, использовав оператор LET, а также провести проверочный расчет по любой сложной формуле с помощью этого оператора.

Переход в непосредственный режим можно осуществить как по воле пользователя, так и по требованию БЕЙСИК-системы при обнаружении ошибки или неясности. Пользователь может перевести систему в непосредственный режим, поместив специальный оператор STOP в нужном месте программы.

Можно также воспользоваться командой с клавиатуры СУ/С во время исполнения программы. Операторы STOP следует помещать в такие точки программы, в которых удобно контролировать правильность ее работы.

После выдачи сообщения

```
STOP AT LINE m  (остановка на строке m)
READY
```

(где m — номер строки с оператором STOP) и останова можно приступить к отладке программы. Для исследования значений конкретных переменных удобно пользоваться оператором PRINT в непосредственном режиме, а для контроля правильности текста программы можно воспользоваться системой команд LIST.

Для выхода из непосредственного режима и перехода к продолжению программы можно воспользоваться оператором GOTO m1 в непосредственном режиме. В качестве метки m1 нужно указать метку оператора, с которого начнется продолжение программы.

БЕЙСИК-система осуществляет проверку операторов программы и вводимых данных и для каждой обнаруженной ошибки печатает соответствующее сообщение об ошибке. В зависимости от версии БЕЙСИК-системы сообщения об ошибках могут иметь различные форматы. Для БЕЙСИК-системы ДВК они могут быть двух видов в зависимости от того, где обнаружена ошибка. Если ошибка обнаружена в программе, формат сообщения об ошибке имеет следующий вид:

? ⟨*текст*⟩ AT LINE m

где ⟨*текст*⟩ — текст, указывающий причину возникновения ошибки; AT LINE — служебное слово (*в строке*); m — номер строки, в которой допущена ошибка.

Если ошибка обнаружена в операторе, выполняемом в непосредственном режиме либо в системной команде, формат сообщения об ошибке:

? ⟨*текст*⟩

где ⟨*текст*⟩ — текст, указывающий причину возникновения ошибки; В обоих случаях ⟨*текст*⟩ печатается на английском языке. Сообщения об ошибках и их расшифровку для БЕЙСИК-системы ДВК см. в приложении 3. Правила исправления ошибок рассмотрены ранее.

Для БЕЙСИК-системы MSX, а также для БЕЙСИК-систем ЭВМ «Электроника-60» и «Электроника 100-25» сообщения об ошибках имеют следующий формат:

ОШИБКА XXX В СТРОКЕ m

где XXX — код ошибки; m — номер строки, в которой произошла ошибка.

Если m = 0, это означает, что ошибка произошла в непосредственном режиме.

Ошибки, обнаруженные в программном режиме, могут быть неустранимыми (аварийными) или предупредительными. После обнаружения аварийной ошибки выполнение программы прекращается и на пульт оператора выдается сообщение об ошибке.

При обнаружении предупредительной ошибки на пульт оператора выдается сообщение об ошибке, ошибочный оператор пропускается и программа продолжает выполняться.

Коды ошибок и причины их возникновения для БЕЙСИК-систем ЭВМ «Электроника-60» и «Электроника 100-25» приведены в приложениях 4 и 5, а для БЕЙСИК-системы MSX — в приложении 6.

Ошибки в программе, вызванные неправильным ее составлением (искажение смысла программы при соблюдении формальных правил записи операторов), в большинстве случаев не отслеживается БЕЙСИК-системой. Следовательно, при этом не выдается сообщений об ошибках. Обнаружение таких ошибок и их устранение полностью возлагается на пользователя.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Какие действия могут выполняться в командном режиме?
2. Каковы основные системные команды и их назначение?
3. Какие команды подаются с помощью специальных клавиш?
4. Как можно исправить ошибочные операторы программы?
5. Какие приемы применяются при отладке программ?
6. Каким образом БЕЙСИК-система реагирует на допущенные ошибки?
7. Как можно вставить пропущенный оператор в программу?
8. Каким образом удалить строку из программы?
9. Как расшифровать сообщение об ошибке?

Приложение 1. Операторы языка БЕЙСИК

Оператор	Назначение оператора
CLOSE	Закрывает файл, т. е. разрывает связь между файлом на ВЗУ и файлом в программе
DATA	Организует блок данных для считывания оператором READ
DEF	Определяет функцию пользователя FN
DIM	Резервирует место в памяти для числовых или строковых массивов
DIM #	Определяет файл как файл с прямым доступом
END	Определяет конец программы
FOR	Указывает начало цикла и определяет его параметры
GOSUB	Осуществляет переход к первому оператору подпрограммы
GO TO	Осуществляет переход к указанной строке
IF THEN IF GO TO	Передает управление в зависимости от истинности или ложности выражения отношения
IF END#THEN IF END#GO TO	Передает управление при обнаружении признака конца файла с последовательным доступом
INPUT	Вводит данные с терминала в процессе выполнения программы
INPUT #	Считывает файл, открытый оператором OPEN FOR INPUT
LET	Присваивает значение переменной
LINPUT	Вводит строковые данные с терминала
LINPUT #	Считывает строковые данные из файла, открытого оператором OPEN FOR INPUT
NEXT	Указывает конец цикла, организованного оператором FOR
ON GOSUB	Передает управление одной из нескольких подпрограмм
ON GOTO	Передает управление одной из указанных строк, в зависимости от значения числового выражения
OPEN	Связывает файл на ВЗУ с файлом в программе
PRINT	Выводит данные на терминал
PRINT USING	Выводит данные на терминал по заданному формату
PRINT #	Помещает данные в файл на ВЗУ
RANDOMIZE	Изменяет начальное значение случайного числа при выполнении функции RND
READ	Считывает данные из блока данных, организованных оператором DATA
REM	Используется для ввода примечаний и комментариев в программу пользователя
RESET	Эквивалентен оператору RESTORE
RESTORE	Служит для восстановления данных в блоке данных, организованном оператором DATA
RETURN	Возвращает управление оператору, следующему за выполненным оператором GOSUB
STOP	Останавливает программу. Ее работа может быть продолжена оператором GO TO

Оператор	Назначение оператора
Операторы графики	
CIRCLE	Строит дуги, окружности, эллипсы
CLS	Очищает экран дисплея
COLOR	Задает цвет изображения на экране
LINE	Строит отрезки прямых линий и прямоугольники
PSET	Строит отдельную точку с заданными координатами
SCREEN	Включает режим графики

Приложение 2. Системные команды БЕЙСИК-системы ДВК

Команда	Назначение команды
BYE	Организует выход из БЕЙСИК-системы
CLEAR	Очищает числовые и строковые переменные, массивы
COMPILE	Компилирует программу и сохраняет ее на устройстве файловой структуры
DEL	Удаляет из программы строки
LIST	Выводит на терминал программу, находящуюся в памяти ЭВМ
NEW	Очищает память и присваивает имя программе, которая будет вводиться
OLD	Помещает в оперативную память программу или данные, находящиеся на ВЗУ
RENAME	Изменяет имя текущей программы на заданное
REPLACE	Заменяет файл, имеющий определенное имя на ВЗУ, другим файлом с таким же именем
RESEQ	Перенумеровывает строки программы, находящейся в памяти
RUN	Запускает на выполнение программу, находящуюся в оперативной памяти, либо помещает в память программу, хранящуюся в файле на ВЗУ, и запускает ее на выполнение, если указано имя файла
SAVE	Сохраняет программу, находящуюся в памяти, на одном из внешних носителей
SCR	Очищает содержимое оперативной памяти
SUB	Изменяет часть строки
UNSAVE	Удаляет файл с ВЗУ
RU BAS	Помещает в память ЭВМ БЕЙСИК-систему, после чего пользователь может работать с системой

Приложение 3. Сообщения об ошибках БЕЙСИК-системы ДВК

Текст сообщения системы	Причина возникновения ошибки
? ARGUMENT ERROR	Аргументы в вызове функции не соответствуют числу или типу аргументов, определенных для данной функции

Текст сообщения системы	Причина возникновения ошибки
? ARRAY TOO LARGE	Объем памяти мал для массива, указанного в операторе DIM
? BAD DATA READ	Элементы данных, вводимых оператором DATA, не соответствуют типу данных в операторе READ
? BAD DATA = RETYPE FROM	Данные, вводимые оператором INPUT, не соответствуют типу данных, указанных в этом операторе
? BAD LOG	Аргумент функции LOG или LOG10 равен нулю или отрицательному значению
? BUFFER STORAGE OVERFLOW	Объем программы пользователя превышен
? CANNOT DELETE FILE	Несуществующий файл указан в команде UNSAVE
? CHANNEL ALREADY OPEN	Оператор OPEN открывает один из двенадцати программных файлов, который был уже открыт
? CHANNEL I/O ERROR	Ошибка ввода — вывода
? CHANNEL NOT OPEN	Программа выполняет операцию ввода — вывода в файл, который предварительно не был открыт
? DIVISION BY ZERO	Деление на нуль
? END NOT LAST	Оператор END не последний в программе
? ERROR CLOSING CHANNEL	Ошибка, возникшая при работе оператора CLOSE
? EXCESS INPUT IGNORED	При работе оператора INPUT вводится больше данных, чем требуется
? EXPONENTIATION ERROR	Неверные аргументы для функции EXP
? FILE ALREADY EXISTS	Оператор OPEN FOR INPUT открывает открытый файл
? FILE NOT FOUND	Требуемый файл отсутствует на определенном устройстве
? FLOATING OVERFLOW	Переполнение в результате вычислений. БЕЙСИК присваивает значение 0 и продолжает выполнение
? FLOATING UNDERFLOW	Значение результата вычислений меньше 10^{-38}. БЕЙСИК присваивает значение 0 и продолжает вычисление
? FOR WITHOUT NEXT	Программа содержит оператор FOR без соответствующего оператора NEXT
? FUNCTION ALREADY DEFINED	Функция пользователя FN определена повторно
? ILLEGAL CHANNEL NUMBER	Указатель файла не входит в диапазон 1—12
? ILLEGAL DEF	Недопустимый формат оператора DEF
? ILLEGAL DIM	Ошибка в операторе DIM
? ILLEGAL IN IMMEDIATE	Обращение к оператору INPUT в непосредственном режиме
? ILLEGAL I/O DIRECTION	Попытка записи в файл, открытый для чтения и наоборот
? INCONSISTENT NUMBER OF SUBSCRIPTS	Обращение к двумерному массиву как к одномерному и наоборот
? INPUT STRING ERROR	В символьной строке, вводимой оператором INPUT, отсутствуют вторые кавычки

Текст сообщения системы	Причина возникновения ошибки
? INTEGER OVERFLOW	Переполнение в результате вычислений с целыми числами
? LINE TOO LONG	Символьная строка больше допустимой в языке БЕЙСИК
? NEGATIVE SQUARE ROOT	Попытка извлечь квадратный корень из отрицательного числа. БЕЙСИК присваивает значение 0 и продолжает вычисление
? NESTED FOR STATEMENTS WITH SAME CONTROL VARIABLE	Наличие двух или более вложенных операторов FOR с одной и той же переменной цикла
? NEXT WITHOUT FOR	Программа содержит оператор NEXT без соответствующего оператора FOR
? NUMBER AND STRINGS	Попытка выполнить действия с числовыми и строковыми выражениями совместно
? OUT OF DATA	Недостаточное количество данных в операторе DATA для READ
? PROGRAM TOO BIG	Введенная программа не помещается в отведенный объем памяти
? RETURN WITHOUT GOSUB	Попытка выполнить оператор RETURN до выполнения GOSUB
? STRING STORAGE OVERFLOW	Недостаточна память для хранения программы
? STRING TOO LONG	Символьная строка превышает 255 символов
? SUBSCRIPT OUT OF BOUNDS	Значение вычисленного индекса вне границы, указанной в операторе DIM, или меньше нуля
? SUBSTITUTE ERROR	Неверный формат оператора SUB
? SYNTAX ERROR	Команда или оператор введены с нарушением правил записи, т. е. не соответствуют формату оператора или команды
? TOO MANY GOSUB	Число вложенных операторов GOSUB превысило 20
? UNDEFINED FUNCTION	Функция пользователя FN неопределенна
? UNDEFINED LINE NUMBER	Неопределенный номер строки
? USE REPLACE	Попытка сохранить уже имеющийся файл командой SAVE

Приложение 4. **Сообщения об ошибках («Электроника-60», «Электроника 100-25»)**

Код ошибки	Причина возникновения ошибки
0	Переполнение памяти, отведенной пользователю
1	Нераспознаваемый оператор
2	Недопустимый оператор GO TO или GOSUB
3	Недопустимый знак, ограничивающий оператор (обычно в случае неправильно сформированного оператора, который вызывает преждевременное окончание действия оператора)
4	Оператор RETURN без соответствующего оператора GOSUB
5	Неправильно сформирован индекс

Код ошибки	Причина возникновения ошибки
6	Индекс не попадает в интервал от 0 до 255 или превышает максимум, установленный программой
7	Несоответствие скобок в операторе
8	Недопустимый оператор
9	Недопустимый знак операции отношения в операторе IF
10	Недопустимый оператор IF
11	Недопустимый оператор PRINT
12	Слишком длинная вводимая строка (превышает 80 знаков)
13	Неправильная размерность в операторе DIM
14	В памяти недостаточно места для массива
15	Неправильно сформирован оператор DEF
16	Недопустимый номер строки или неправильное значение размерности
17	Оператор DIM для ранее описанного и использованного элемента
18	Неправильная переменная в списке оператора INPUT
19	Неправильная переменная в списке оператора READ
20	Данные в списке оператора READ исчерпаны
21	Неправильный формат оператора DATA
22	Недопустимый оператор FOR
23	FOR не сопровождается соответствующим оператором NEXT
24	Оператор NEXT без оператора FOR
25	Несоответствие кавычек в операторе
26	Неправильно установлена функция EXF
27	Неправильно сформировано выражение (пропущен порядок числа в формате E)

Приложение 5. **Предупредительные сообщения об ошибках («Электроника-60», «Электроника 100-25»)**

Код ошибки	Причина возникновения ошибки
120	Недопустимые знаки при вводе
121	Введено недостаточно данных по оператору INPUT
122	Введено слишком много данных по оператору INPUT
123	Несуществующая переменная
124	Число слишком велико для фиксации (вероятно, комбинация индексов превышает диапазон)
125	Переполнение и заем при делении/умножении
126	Квадратный корень из отрицательного числа
127	Логарифм отрицательного числа или нуля; переполнение при вычислении

Код ошибки	Причина возникновения ошибки
1	NEXT без FOR NEXT не предшествовал FOR или переменная, использованная в FOR, не соответствует переменной, использованной NEXT
2	Синтаксическая ошибка Неверное использование символов, например число открывающих скобок не соответствует числу закрывающих; неправильная запись операторов или их составных частей; неправильно использованная запятая и т. п.
3	RETURN без GOSUB При выполнении RETURN обнаружено, что не было выполнено GOSUB
4	Исчерпан список оператора DATA При выполнении оператора READ обнаружено, что список оператора DATA исчерпан
5	Недопустимое использование функции
6	Переполнение Результат арифметической операции слишком велик и не может быть записан в формате, принятом для чисел в БЕЙСИК-системе. В случае очень малого значения порядка результат приравнивается нулю
7	Переполнение памяти Программа не умещается в памяти или использовано слишком много операторов FOR или переменных
8	Не определен номер строки В операторах, содержащих ссылку на номер строки (например, GO TO, GOSUB, IF... THEN... ELSE), использован несуществующий номер строки
9	Недопустимый индекс Ссылка на элемент массива с индексом, который выходит за пределы размерности массива, объявленного в операторе DIM, либо указано неправильное число индексов
10	Повторное определение массива Массив определен двумя операторами DIM или массив определен оператором DIM после того, как по умолчанию для этого массива была установлена размерность 10
11	Деление на нуль В выражении встретилось деление на нуль или нуль был возведен в отрицательную степень
12	Недопустимый оператор или команда в режиме непосредственного выполнения
13	Ошибка в типах Попытка присвоить символьной переменной числовое значение или наоборот. Либо функции, использующей числовой аргумент, передается символьный аргумент или наоборот
14	Не хватает места символьным переменным Превышено количество ячеек памяти, которая была отведена символьным переменным
15	Слишком длинная строка Была сделана попытка создать строку длиной более чем 255 символов
16	Слишком длинное или сложное символьное выражение Его следует разбить на более простые
17	Продолжение выполнения программы невозможно Была сделана попытка продолжить выполнение программы, ко-

Код ошибки	Причина возникновения ошибки
	торая: 1) была прервана из-за возникновения ошибки; 2) изменена во время останова; 3) не существует
18	Не определена функция пользователя Попытка обратиться к функции до ее определения оператором DEF
19	Ошибка устройства ввода — вывода Встречается при работе устройств ввода — вывода
20—23	Не определены
24	Отсутствует операнд Выражение содержит оператор без операнда; команда или оператор заданы без обязательных параметров
25	Переполнение буфера ввода Была сделана попытка ввести строку длиной более чем 255 символов
26—51	Не определены
52	Ошибочный номер файла Оператор или команда ссылается на файл, который не открыт или номер файла выходит за первоначально определенный предел номеров файлов
53	Файл не найден
54	Файл уже открыт Применен оператор OPEN для файла, который уже открыт
55	Попытка прочитать запись из пустого файла Оператор INPUT # был выполнен после введения из файла всех данных или применен для пустого файла. Чтобы избежать этой ошибки, для обнаружения окончания файла применять функцию EOF
56	Ошибочное имя файла В операторах использовано неправильное имя файла, либо оно состоит из слишком большого количества символов
57	Команда непосредственного выполнения в программе, находящейся на диске или на кассете во время загрузки файла. Загрузка заканчивается
58	Попытка неправильного обращения к файлу
59	Файл не открыт Команда (или оператор) ввода — вывода была применена к файлу, который не был открыт
60—61	Не определены
62	Ошибочное имя устройства
63—255	Резервные коды ошибок

Десятичный код	Обозначение	Наименование
2	НТ	Начало текста: CTRL/B
3	КТ	Конец текста: CTRL/C
8	—	Возврат на шаг: CTRL/H
9	ГТ	Горизонтальная табуляция: CTRL/I (TAB)
10	ПС	Перевод строки: CTRL/J
11	ВТ	Вертикальная табуляция: CTRL/K
12	—	Очистка экрана: CTRL/L
13	ВК	Возврат каретки: CTRL/M (RETURN)
15	ВХ	Вход: CTRL/O
19	—	Управление устройством: CTRL/S
21	—	Отрицание: CTRL/U
32	ПР	Пробел: клавиша пропуска
33	!	Восклицательный знак
34	"	Кавычки
35	#	Номер
36	¤	Знак денежной единицы
37	%	Процент
38	&	Коммерческое И (амперсанд)
39	'	Апостроф
40	(	Круглая скобка левая (открывающая)
41	)	Круглая скобка правая (закрывающая)
42	*	Звездочка
43	+	Плюс
44	,	Запятая
45	—	Минус
46	.	Точка
47	/	Дробная черта
48	0	Цифры от 0 до 9
49	1	
50	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	
58	:	Двоеточие
59	;	Точка с запятой
60	<	Меньше
61	=	Равно
62	>	Больше
63	?	Вопросительный знак
64	@	Коммерческое ЭТ
65	A	С этого знака до Z используется нижний регистр
66	B	
67	C	
68	D	Прописные буквы латинского алфавита
69	E	
70	F	
71	G	
72	H	
73	I	
74	J	
75	K	
76	L	
77	M	
78	N	
79	O	
80	P	
81	Q	
82	R	
83	S	
84	T	
85	U	
86	V	
87	W	
88	X	
89	Y	
90	Z	
91	[	Квадратная скобка открывающая
92	\	Черта с левым наклоном
93	]	Квадратная скобка закрывающая
94	↑	Стрелка вверх
95	—	Знак подчеркивания
96	Ю	С этого знака до буквы Ч используется верхний регистр
97	А	
98	Б	
99	Ц	Буквы русского алфавита
100	Д	
101	Е	
102	Ф	
103	Г	
104	Х	
105	И	
106	Й	
107	К	
108	Л	
109	М	

Десятичный код	Обозначение	Наименование	Десятичный код	Обозначение	Наименование
110	Н		117	У	
111	О		118	Ж	
112	П		119	В	
113	Я		120	Ь	
114	Р		121	Ы	
115	С		122	З	
116	Т				

СПИСОК ЛИТЕРАТУРЫ

1. *Кетков Ю. Л.* Программирование на БЕЙСИКе. — М.: Статистика, 1978.

2. *Уорт Т.* Программирование на языке БЕЙСИК: Пер. с англ./Под ред. *В. Ф. Шаньгина*. — М.: Машиностроение, 1982.

3. Программирование на языке БЕЙСИК-ПЛЮС для СМ-4. — М.: Финансы и статистика, 1982.

4. *Гиглавый А. В., Гнездилова Г. Г., Гуткин М. Л.* Программирование на языке БЕЙСИК для школьной ЭВМ. — М.: Вычислительный центр АН СССР, 1986.

5. *Гиглавый А. В., Гнездилова Г. Г., Гуткин М. Л.* Дополнительные возможности языка БЕЙСИК для школьной ЭВМ. — М.: Вычислительный центр АН СССР, 1986.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МИКРОЭВМ

В 11-ти книгах

Владимир Федорович Шаньгин,
Анатолий Яковлевич Пьянзин

Кн. 5. ДИАЛОГОВЫЙ ЯЗЫК БЕЙСИК

Заведующая редакцией Н. И. Хрусталева
Редактор И. Е. Якушина
Художник Ю. В. Дьяконов
Художественный редактор В. И. Мешалкин
Технические редакторы Т. А. Новикова, З. В. Нуждина
Корректор Р. К. Косинова

ИБ № 6686

Изд. № Стд—568. Сдано в набор 19.05.87. Подп. в печать 08.10.87. Т-20913. Формат $60\times90^{1}/_{16}$. Бум кн.-журн. Гарнитура литературная. Печать офсетная. Объем 7 усл. печ. л. 7,5 усл. кр.-отт. 6,75 уч. изд. л. Тираж 100000 экз. Зак. № 664. Цена 35 коп.
Издательство «Высшая школа», 101430, Москва, ГСП-4, Неглинная ул., д. 29/14.

Ярославский полиграфкомбинат Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 150014, Ярославль, ул. Свободы, 97.