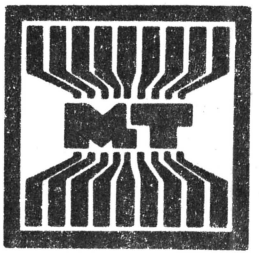

Программирование на Бейсике

Последний пример, который мы рассмотрим, посвящен разработке программ, организующих вывод на экран дисплея графических изображений. Формирование рисунков, состоящих из отрезков прямых линий и отдельных точек, не вызывает обычно каких-либо осложнений — для этого служат графические операторы Бейсика (CLS, PLOT, LINE). Необходимо только постоянно следить за тем, чтобы координаты, задаваемые этим операторам, не выходили за допустимые пределы, иначе интерпретатор выдаст сообщение об ошибке номер 08.

Формирование изображений окружностей и эллипсов — более сложная задача. Приводимая в табл. 5 универсальная подпрограмма позволяет «начертить» окружность произвольного радиуса в любом месте экрана. Эту подпрограмму можно включить в состав вашей программы и обращаться к ней при помощи оператора GOSUB 9000. Вообще говоря, так же, как и в случае с библиотекой подпрограмм в машинных кодах, о которой мы говорили выше, вы можете создать и библиотеки подпрограмм на Бейсике. Например, к описываемой ниже подпрограмме можно добавить подпрограммы для формирования изображения и других геометрических фигур. В этом случае основная

Таблица 5

```
9000 REM ****************************
9010 REM * ПОДПРОГРАММА ВЫЧЕРЧИВАНИЯ *
9020 REM * ИЗОБРАЖЕНИЙ ОКРУЖНОСТЕЙ И *
9030 REM * ЭЛЛИПСОВ.                 *
9040 REM ****************************
9050 P2=3.14159*2: REM УГОЛ = 2 ПИ
9060 REM ДЛЯ ВЫЧЕРЧИВАНИЯ ДУГ НАДО
9070 REM ИЗМЕНИТЬ КОНСТАНТЫ В ОПЕРАТОРЕ
9080 REM ИНИЦИАЛИЗАЦИИ ЦИКЛА
9100 FOR F9=0 TO P2 STEP 0.1
9110 X9 = XC + RC*KX*SIN(F9)
9120 Y9 = YC + RC*KY*COS(F9)
9130 IF X9<0 THEN X9=0
9140 IF X9>127 THEN X9=127
9150 IF Y9<0 THEN Y9=0
9160 IF Y9>63 THEN Y9=63
9170 PLOT X9,Y9,FL
9180 NEXT F9
9190 RETURN
```

программа будет состоять из операторов присваивания, подготавливающих параметры для подпрограмм и операторов вызова подпрограмм. В Бейсике передача параметров к подпрограммам происходит присвоением значений соответствующим переменным. Все переменные в программе на Бейсике — глобальные. Это значит, что если значение переменной изменилось при выполнении подпрограммы, то это отразится и на выполнении основной программы. Поэтому программист должен следить за сохранением значения переменных, необходимых для работы основной программы.

Для подпрограмм, входящих в библиотеку, принято назначать большие номера строк. Поэтому при написании новой программы сначала загрузите с магнитной ленты вашу «библиотеку», а основную программу набирайте со строки с номером 10. И еще один совет. В качестве подпрограмм целесообразно оформлять только часто выполняемые процедуры, с остальными этого лучше не делать, так как вызов подпрограммы и возврат из нее требуют определенных затрат времени.

Перед обращением к подпрограмме (табл. 5) необходимо задать значения переменным, с которыми она работает. Переменные XC и YC определяют координаты центра окружности, а RC — ее радиус. Переменными KX и KY задают степень сжатия или растяжения по осям X и Y соответственно. Изменяя их значения, можно получать изображения окружности, вытянутые вдоль оси X или Y. При KX и KY, равных 1, на экране формируется эллипс, вытянутый вдоль оси Y. Это объясняется особенностями телевизионного растра и нашего дисплейного модуля. Поэтому для получения изображения окружности параметр KY должен быть равен $\approx 0{,}8KX$.

В подпрограмме для вычисления координат точек, лежащих на окружности заданного радиуса R, использованы известные соотношения:

$X = XC + RC * \sin(\varphi)$,
$Y = YC + RC * \cos(\varphi)$,

где XC и YC — координаты центра окружности.

В подпрограмме предусмотрены контроль и коррекция координат центра окружности по следующим правилам:

если $XC < 0$, то $XC = 0$;
если $XC > 127$, то $XC = 127$;
если $YC < 0$, то $YC = 0$;
если $YC > 63$, то $YC = 63$.

Поэтому при выполнении подпрограммы сообщение об ошибке 08 не возникает.

Таблица 6

```
10 REM ****************************
20 REM * ПРОГРАММА ФОРМИРОВАНИЯ    *
30 REM * ИЗОБРАЖЕНИЯ ОЛИМПИЙСКОГО *
40 REM * СИМВОЛА. ИСПОЛЬЗУЕТ ПОД- *
50 REM * ПРОГРАММУ "КРУГ"          *
60 REM ****************************
70 CLS
80 KX=1:KY=0.8:REM КОЭФФИЦИЕНТЫ СЖАТИЯ
90 RC=10:REM РАДИУС ОКРУЖНОСТЕЙ
100 YC=40
110 XC=40:GOSUB 9000:REM 1 КРУГ
120 XC=56:GOSUB 9000:REM 2 КРУГ
130 XC=72:GOSUB 9000:REM 3 КРУГ
140 YC=25
150 XC=48:GOSUB 9000:REM 4 КРУГ
160 XC=64:GOSUB 9000:REM 5 КРУГ
170 STOP
```

В табл. 6 приведена программа, формирующая на экране изображение олимпийского символа — пяти пересекающихся колец, в которой использована описанная подпрограмма.

ПРОГРАММА И ПАМЯТЬ ЭВМ

При разработке программ на любом языке программирования практически всегда приходится учитывать такие факторы, как время выполнения программы и необходимый для этого объем памяти. Оба параметра следует по возможности уменьшать, это позволит и результат получить быстрее, и учесть ограничения, связанные с небольшим объемом памяти ЭВМ. Однако добиться одновременно и того, и другого — довольно сложная задача.

К программам на Бейсике сказанное имеет самое прямое отношение, особенно если транслятор с языка высокого уровня (как в нашем случае) реализован в виде интерпретатора и объем памяти для хранения программ и переменных мал. Рассмотрим несколько приемов, позволяющих уменьшить требуемый для работы программ объем памяти.

Окончание. Начало см. в «Радио», 1985, № 2, с. 34.

При работе интерпретатора в ОЗУ одновременно находятся сам интерпретатор, текст программы на Бейсике и различные переменные и константы, встречающиеся в программе. Появление сообщения об ошибке 07, как при наборе текста программы, так и при ее выполнении, указывает на то, что для данной программы объем памяти недостаточен. Как следует поступать в таких случаях? Прежде всего можно сократить в программе комментарии, уменьшить длину текстовых сообщений, использовать однобуквенные имена для переменных. Объем памяти, занимаемый текстом программы, можно также уменьшить, набирая в каждой строке не по одному, а по несколько операторов, отделяя их друг от друга символом «:» — двоеточием (почему происходит такое сокращение — вы поймете, познакомившись с дополнительными сведениями об интерпретаторе).

Если после этих переделок объем памяти все равно недостаточен, то следует критически проанализировать необходимость использования тех или иных переменных и особенно переменных с индексами — массивов. Для массивов следует резервировать ровно столько ячеек памяти, сколько для них в действительности потребуется. Например, если массив A(I) состоит из 5 элементов, а вы его не описали оператором DIM, то по умолчанию интерпретатор выделит в памяти место для 10 элементов массива, и это приведет к потере 20 байт памяти. Для того чтобы оценить, к каким затратам памяти приводит тот или иной вариант реализации алгоритма, удобно пользоваться встроенной функцией FRE (0). Попробуйте посмотреть, чему равно значение этой функции до загрузки программы в память, после загрузки и после выполнения программы. Для этого достаточно в непосредственном режиме набрать PRINT FRE (0).

Таким образом, вы сможете определить, сколько места в памяти занимает текст программы и сколько ячеек отводится для хранения переменных и констант.

ПРОГРАММА И ВРЕМЯ ЕЕ ВЫПОЛНЕНИЯ

Сократить время выполнения программы несколько сложнее, чем ее объем. Для этого прежде всего необходимо определить ту часть программы, на выполнение которой затрачивается наибольшее время.

Известно, что циклическая часть программы, занимающая всего 5% исходного текста, требует для своего выполнения обычно почти 95% времени работы всей программы. Вы должны обнаружить эти «критические» циклы в программе и попытаться уменьшить время их выполнения. Вначале надо убедиться, что все операторы, входящие в этот цикл, действительно должны выполняться внутри него. Например, если значение переменной не изменяется в цикле (см. табл. 7, пример 1), то ее инициализацию следует производить вне цикла (пример 2), что сократит время его выполнения. Уменьшить время выполнения программы можно также некоторым видоизменением операторов, входящих в тело цикла. В примере 3 (табл. 7) обнуление элементов массива проводится в цикле. Если переписать эту программу, как показано в примере 4, то время выполнения операторов FOR.....NEXT значительно сократится.

Таблица 7

```
10 REM ************
20 REM * ПРИМЕР 1 *
30 REM ************
40 FOR I=0 TO 340
50 A=SIN(1.45)*30
60 B(I)=A*I/4
70 NEXT I
80 STOP
```

```
10 REM ************
20 REM * ПРИМЕР 2 *
30 REM ************
40 A=SIN(1.45)*30
50 FOR I=0 TO 340
60 B(I)=A*I/4
70 NEXT I
80 STOP
```

```
10 REM ************
20 REM * ПРИМЕР 3 *
30 REM ************
40 FOR I=1 TO 60
50 S(I)=0
60 NEXT I
70 STOP
```

```
10 REM ************
20 REM * ПРИМЕР 4 *
30 REM ************
40 FOR I=1 TO 30
50 S(I)=0:S(I+30)=0
60 NEXT I
70 STOP
```

Во многих случаях, если число элементов массива невелико, имеет смысл вообще отказаться от оператора цикла и проводить инициализацию элементов массива следующим образом:

$$10A(1)=\emptyset:\ A(2)=\emptyset:\ A(3)=\emptyset:\ A(4)=\emptyset:\ A(5)=\emptyset.$$

При вычислении результатов выражений имеется возможность сократить время вычислений благодаря использованию дополнительных переменных. Например, при выполнении строки IF K*R/T—D>∅ THEN E=K*R/T+1 дважды определяется величина K*R/T. Если переписать эту строку следующим образом:

Q=K*R/T: IF Q—D>∅ THEN E=Q+1,

то вычисление значения новой переменной Q будет проведено только один раз. «Плата» за сокращение времени работы программы — необходимость резервирования нескольких ячеек памяти для хранения переменной Q.

Для обращения к элементу массива интерпретатору, как правило, требуется больше времени, чем для обращения к простой переменной. Это связано с необходимостью обработки индексов. Поэтому рекомендуется всегда, если это возможно, использовать простые переменные, которые будут хранить значения часто используемых элементов массива. Например, элементы массива

$$Z1=Y(A)+10:\ Z2=Y(A)-5:\ Z3=Y(A)+20$$

целесообразнее определять так:

$$Z1=Y(A)+10:\ Z2=Z1-15:\ Z3=Z1+10.$$

Особенно большой выигрыш во времени может быть получен, если подобные вычисления проводятся в теле цикла или в программе использованы многомерные массивы.

Мы рассмотрели ряд конкретных примеров, позволяющих сократить время выполнения программ на Бейсике, но, конечно, наиболее ощутимые результаты можно получить, если воспользоваться функцией USR (X) и запрограммировать критичный по времени выполнения фрагмент алгоритма на языке Ассемблер.

Таким образом, перед программистом всегда стоит проблема выбора наилучшего варианта реализации программы с точки зрения времени ее выполнения, занимаемого объема памяти, простоты и ясности.

ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ ОБ ИНТЕРПРЕТАТОРЕ

Посмотрим теперь, как хранится программа на Бейсике в памяти ЭВМ. Это может оказаться полезным в случае какого-либо сбоя в работе микро-ЭВМ, который может привести к потере набранной программы. Кроме того, приводимые ниже сведения помогут восстановить программу, введенную с ошибками с магнитной ленты, и разобраться в организации структуры данных в ваших программах.

На рис. 3 показано, как распределяется память в микро-ЭВМ при работе интерпретатора Бейсика. Текст программы хранится в памяти сразу же после кодов интерпретатора, за ним следует область памяти, выделенная для хранения переменных. Вблизи верхней границы ОЗУ имеется специальная область (стек), отведенная для «внутренних» нужд интерпретатора.

Каким образом строка программы представлена в микро-ЭВМ? Просматривая содержимое памяти с помощью директив «D» и «L» Монитора, вы можете обнаружить, что хранимая в памяти информация не похожа на то, что

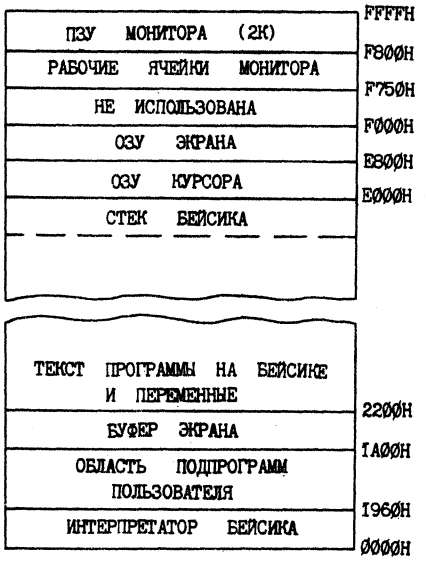

Рис. 3

Таблица 8

ТАБЛИЦА КОДОВ КЛЮЧЕВЫХ СЛОВ БЕЙСИКА

СЛОВО	МЕСТ.	ДЕС.	СЛОВО	МЕСТ.	ДЕС.	СЛОВО	МЕСТ.	ДЕС.
CLS	80	128	CONT	97	151	SGN	AD	174
FOR	81	129	LIST	98	152	INT	AE	175
NEXT	82	130	CLEAR	99	153	ABS	AF	176
DATA	83	131	MLOAD	9A	154	USR	B0	177
INPUT	84	132	MSAVE	9B	155	FRE	B1	178
DIM	85	133	NEW	9C	156	INP	B2	179
READ	86	134	TAB(	9D	157	POS	B3	180
CUR	87	135	TO	9E	158	SQR	B4	181
GOTO	88	136	SPC(	9F	159	RND	B5	182
RUN	89	137	FN	A0	160	LOG	B6	183
IF	8A	138	THEN	A1	161	EXP	B7	184
RESTORE	8B	139	NOT	A2	162	COS	B8	185
GOSUB	8C	140	STEP	A3	163	SIN	B9	186
RETURN	8D	141	+	A4	164	TAN	BA	187
REM	8E	142	-	A5	165	ATN	BB	188
STOP	8F	143	*	A6	166	PEEK	BC	189
OUT	90	144	/	A7	167	LEN	BD	190
ON	91	145	^	A8	168	STR$	BE	191
PLOT	92	146	AND	A9	169	VAL	BF	192
LINE	93	147	OR	AA	170	ASC	C0	193
POKE	94	148	>	AB	171	CHR$	C1	194
PRINT	95	149	=	AC	172	LEFT$	C2	195
DEF	96	150	<	AD	173	RIGHT$	C3	196
						MID$	C4	197

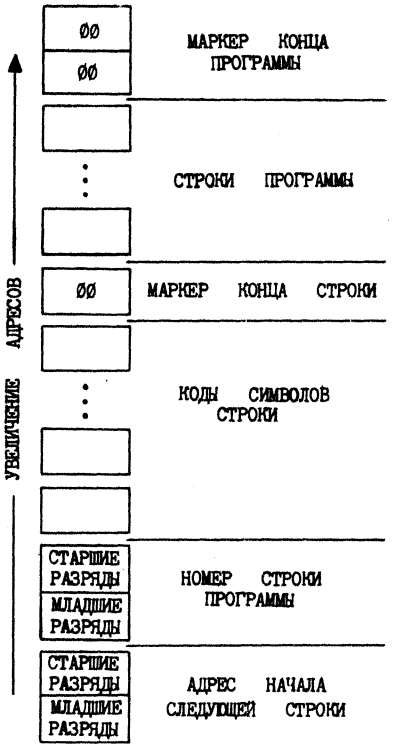

Рис. 4

было набрано на клавиатуре. Это вполне объяснимо, так как микро-ЭВМ «понимает» только двоичные коды, т. е. способна «понять» код той или иной буквы, а не саму букву. Поэтому строки программы хранятся в памяти в виде двоичных кодов и переводятся в символьный вид только в случае просмотра текста программы по директиве интерпретатора LIST. Однако это только одна сторона вопроса. Если бы каждый вводимый символ программы занимал в памяти одну ячейку (1 байт), то это потребовало бы, во-первых, очень большого объема памяти и, во-вторых, программа выполнялась бы очень медленно.

Вместо того, чтобы хранить в памяти коды всех символов исходного текста программы, можно закодировать каждое ключевое слово всего одним байтом. Это вполне возможно, так как из 256 возможных двоичных кодов ($2^8=256$), которые можно записать в одну ячейку памяти, для кодирования алфавитно-цифровых символов используется только 128. Двоичные коды, у которых старший бит равен 1, и использованы для кодирования ключевых слов языка Бейсик (табл. 8). Что же дает такое «сжатие» информации? Очень многое. Например, если в программе 100 раз встречается оператор GOSUB, то кодирование позволяет сэкономить 400 байт памяти!

На рис. 4 показан формат строки программы на Бейсике в том виде, в каком она хранится в памяти микро-ЭВМ. В начале каждой строки два байта отведены для хранения указателя адреса начала следующей строки программы, следующие два байта хранят номер строки, а заканчивается она байтом, заполненным одними нулями. Таким образом, текст программы хранится в памяти в виде специальной структуры данных, называемой в литературе «односвязным списком».

Заканчивая обработку очередной строки программы, интерпретатор последовательно просматривает указатели списка до тех пор, пока не будет найдена строка с требуемым номером. Конец списка помечается двумя «нулевыми» байтами. Вы таким же образом можете вручную (с помощью директив Монитора) определить, где заканчивается программа, просматривая указатели списка до тех пор, пока не обнаружите три смежных байта, заполненных нулями. Во многих практических случаях, воспользовавшись рассмотренными рекомендациями, можно восстановить программу, в которой в результате сбоя была нарушена целостность списка. После восстановления структуры списка необходимо изменить значения, хранящиеся в ячейках памяти 0245H и 0246H. В этих ячейках хранятся значения соответственно младшего и старшего байта конечного адреса программы. Этот адрес на двойку превосходит адрес первого байта маркера конца списка.

Г. ЗЕЛЕНКО,
В. ПАНОВ,
С. ПОПОВ

г. Москва