ИНФОРМАТИКА

Учебное пособие для педагогических институтов

ИНФОРМАТИКА

2	3	4	5	6	7	8	9	A	B
!	0	@	P	`	p	▔	◀	α	≡
"	1	A	Q	a	q	▚	⧗	β	±
#	2	B	R	b	r	▀	⧓	Γ	≥
$	3	C	S	c	s	▁	▪	π	≤
%	4	D	T	d	t	·	▫	Σ	⌠
&	5	E	U	e	u	▌	▪	σ	⌡
'	6	F	V	f	v	▗	▪	μ	÷
(	7	G	W	g	w	▞	▩	τ	≈
)	8	H	X	h	x	▙	▵	Φ	°
*	9	I	Y	i	y	▐	∓	Θ	∙
+	:	J	Z	j	z	▟	∞	Ω	·
,	;	K	[	k	{	▒	■	δ	√
-	<	L	\	l	|	◤	▬	∞	n
.	=	M	]	m	}	◢	▮	∅	2
/	>	N	^	n	~	▲	▮	∈	■

ИНФОРМАТИКА

Утверждено
Государственным комитетом СССР
по народному образованию
в качестве учебного пособия
для педагогических специальностей
высших учебных заведений

МОСКВА «ПРОСВЕЩЕНИЕ» 1991

ББК 73
И74

Авторы:
А. Р. Есаян, В. И. Ефимов, Л. П. Лапицкая, Э. А. Пащенко, Н. М. Добровольский

Рецензенты:
кафедра вычислительной математики ЛГПИ им. А. И. Герцена (зав. кафедрой доцент, кандидат технических наук *Ю. К. Кузнецов*);
доктор физико-математических наук *В. В. Щенников*

Информатика: Учеб. пособие для пед. спец. высш. учеб. И74 заведений/ А. Р. Есаян, В. И. Ефимов, Л. П. Лапицкая и др.— М.: Просвещение, 1991.— 288 с.: ил.— ISBN 5-09-002699-8.

В книге излагается описание языка Бейсик, модели, методы, алгоритмы и программы обработки информации и решения задач. Первая часть книги, ориентируясь на неподготовленного читателя, призвана создать у него необходимый объем понятий по программированию. Вторая часть книги позволяет развить умение применять программирование для решения практических задач и использования компьютеров в учебном процессе.

И $\frac{4309000000-240}{103(03)-91}$ 21—90 **ББК 73**

Учебное издание

Есаян Альберт Рубенович, Ефимов Владимир Иванович, Лапицкая Людмила Петровна, Пащенко Эдуард Александрович, Добровольский Николай Михайлович

ИНФОРМАТИКА

Зав. редакцией *Т. А. Бурмистрова*. Редактор *Н. А. Шагирова*.
Младший редактор *О. В. Котенкова*. Художник *В. В. Костин*.
Художественный редактор *Ю. А. Пахомов*. Технический редактор *М. М. Широкова*.
Корректор *Н. С. Соболева*

ИБ № 12631

Сдано в набор 12.09.89. Подписано в печать 12.10.90. Формат 60 × 90¹/₁₆. Бумага типограф. № 2. Гарнит. литер. Печать высокая. Усл. печ. л. 18 + 0,25 форзац. Усл. кр.-отт. 18,88. Уч.-изд. л. 17,29 + 0,42 форзац. Тираж 176 000 экз. Заказ № 627. Цена 95 к.

Ордена Трудового Красного Знамени издательство «Просвещение» Министерства печати и массовой информации РСФСР. 129846, Москва, 3-й проезд Марьиной рощи, 41.

Саратовский ордена Трудового Красного Знамени полиграфический комбинат Министерства печати и массовой информации РСФСР. 410004, Саратов, ул. Чернышевского, 59.

ISBN 5-09-002699-8

ЧАСТЬ I. ЯЗЫК БЕЙСИК

Бейсик не нуждается в рекламе. Он нужен всем: и профессионалу-программисту, и в еще большей степени — любителю.

Г. Моррил

ВВЕДЕНИЕ

В прошлом веке один английский миссионер, желая облегчить контакт с туземным населением, выделил из английского языка самую простую и распространенную его часть, содержащую около 300 слов и не имеющую почти грамматики. Это подмножество языка, названное BASIC ENGLISH, действительно оказалось весьма простым для усвоения и поэтому вскоре завоевало популярность не только среди туземцев, но и иммигрантов.

Подобную цель создания средства для расширения и облегчения контактов, только не между различными группами людей, а между непрофессионалами и ЭВМ, поставили перед собой сотрудники Дартмутского колледжа Дж. Кемени и Т. Курц. Разработанный ими в 1964 году алгоритмический язык BASIC (Бейсик), как и всякий другой язык программирования, является формальной знаковой системой, используемой для связи человека с ЭВМ и предназначенной для описания данных и алгоритмов их обработки на вычислительной машине. Название BASIC является аббревиатурой английской фразы «*Beginner's All — purpose Symbolic Instruction Code*», что в переводе означает «многоцелевой язык символических команд для начинающих».

Важной чертой Бейсика является не только его простота и доступность, но и предоставляемая им возможность решать задачи в режиме диалога с ЭВМ.

Для многих современных мини- и микроЭВМ Бейсик предназначался в качестве единственного языка программирования высокого уровня. Это обстоятельство привело к появлению различных его версий, включающих в себя многочисленные эффективные средства программирования из других алгоритмических языков. Таким образом, на сегодняшний день мы имеем не конкретный Бейсик, а целую группу однотипных диалоговых языков, называемых этим же именем. В частности, нас будет интересовать версия 1.0 одной из его достаточно мощных и интересных разновидностей, именуемых MSX-Бейсиком. Именно о ней в основном и идет речь ниже. Изложение языка ориентировано на использование персонального MSX-компьютера «Ямаха».

Для дальнейшего изложения нам необходимо иметь предварительное разъяснение некоторых основных понятий Бейсика, которые затем будут расширяться и уточняться. Начнем с примера программы.

```
10 INPUT X
20 LET Y=X+5:LET Z=X*Y
30 PRINT Y, Z
40 END
```

В общем случае программа на Бейсике есть нумерованная последовательность строк. Номер строки задается целым неотрицательным числом, служащим для ее идентификации. За номером располагаются команды (операторы, приказы, инструкции). Если команд в строке несколько, то друг от друга они отделяются символом двоеточия. Каждая команда представляет собой указание компьютеру — что по ней необходимо сделать и над какими данными. Набор допустимых команд строго ограничен. Для успешного использования Бейсика необходимо этот набор изучить. При этом придется обращать внимание как на синтаксис (правила записи), так и на семантику (смысловое значение) всякой команды.

Для выполнения программы она должна быть занесена в оперативную память компьютера. Первый раз запись в память ЭВМ производится с клавиатуры набором ее текста приблизительно так, как это осуществляется при печатании на пишущей машинке. Правда, здесь все значительно проще, ибо имеется возможность не только отслеживать вводимый текст на экране дисплея, но при необходимости и эффективно его редактировать. Заметим, что программная, или, по-другому, логическая, строка может занимать несколько физических строк экрана дисплея. Далее, ввод логической строки всегда должен заканчиваться нажатием клавиши «возврат каретки» (←┘).

Из памяти компьютера программа под каким-нибудь именем через специальные устройства (дисковод, магнитофон) может быть записана для хранения и дальнейшего использования на внешний магнитный носитель (дискета, кассета). При этом отпадает необходимость в повторном вводе такой программы через клавиатуру. Гораздо проще это делать считыванием ее в память с соответствующего носителя.

Нажатием кнопки ПУСК (команда RUN) находящаяся в оперативной памяти программа начинает автоматически выполняться. Делается это так. Сначала фиксируется строка с наименьшим номером и слева направо последовательно друг за другом выполняются ее команды. Затем эта же процедура повторяется со следующей строкой и т. д., пока не встретится команда END или не будет реализована последняя строка с наибольшим номером. Правда, следует иметь в виду, что, как правило, программы содержат так называемые команды передач управления, являющиеся «нарушителями» описанного идиллического порядка вычислений.

Возвращаясь к нашему примеру, можно отметить, что представляемая им программа содержит 4 строки с номерами: 10, 20, 30 и 40. В строке 20 — две команды, а в остальных — по одной. Дадим описание этих команд.

1) INPUT X — команда ввода данных с клавиатуры.

По команде INPUT (ввод) происходит приостановка вычислений. На экране дисплея появляется знак вопроса. Если ввести с клавиатуры число, то оно станет значением переменной X, и вычисления в программе будут продолжены.

2) LET Y=X+5 — команда присваивания значений.

По команде LET (пусть) организуется вычисление выражения X+5 при текущем значении переменной X. Полученное значение присваивается переменной Y.

3) LET Z=X*Y — команда присваивания значений.

По команде LET организуется вычисление выражения X*Y при текущих значениях переменных X и Y. Полученное значение присваивается переменной Z. В записи команд присваивания служебное слово LET можно опускать.

4) PRINT Y, Z — команда вывода данных.

По команде PRINT на экране дисплея индицируются текущие значения переменных Y и Z.

5) END — команда завершения вычислений.

Теперь мы в состоянии проследить за значениями переменных величин программы (1) при ее конкретном выполнении. Например, если по команде INPUT ввести для X значение 3, то на экран будут выведены числа 8 и 24.

Остановимся еще на одном обстоятельстве. Любая строка Бейсика, вводимая в память без номера, исполняется сразу. Подобные «непосредственные» вычисления допустимы, но носят вспомогательный характер.

Более полное изложение синтаксиса и семантики упомянутых выше команд будет дано в последующих разделах пособия. Сейчас же отметим, что при описании конструкций Бейсика для указания необходимости выбора одного из нескольких возможных элементов последние будем располагать столбиком и заключать в фигурные скобки {}. Далее, необязательные элементы конструкций будем записывать в квадратные скобки [ ].

1. ОСНОВЫ ЯЗЫКА БЕЙСИК И ПРОГРАММИРОВАНИЯ

1.1. Основные объекты языка

Алфавит. Основными объектами, используемыми в языке Бейсик, являются константы, переменные, выражения, функции и массивы. Каждый из этих объектов характеризуется именем (идентификатором), значением и типом принимаемых значений. Имена и значения строятся из фиксированного набора символов, или, по-другому, литер. Совокупность используемых литер, в которую входят прописные и строчные буквы русского и латинского алфавитов, арабские цифры и ряд дополнительных символов (см. таблицу 1), а также около 200 служебных слов (см. приложение 1), и составляют алфавит Бейсика.

Таблица 1

Шестнадцатиричные коды ASCII некоторых символов Бейсика

q \ p	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		0	@	P	`	p	_	◀	α	≡	ю	п	Ю	П
1	!	1	A	Q	a	q	↰	Σ	℘	±	а	я	А	Я
2	"	2	B	R	b	r	■	Ж	Г	≥	б	р	Б	Р
3	#	3	C	S	c	s	‾	▪	π	≤	ц	с	Ц	С
4	$	4	D	T	d	t	·	▪	Σ	ſ	д	т	Д	Т
5	%	5	E	U	e	u	■	▪	σ	J	е	у	Е	У
6	&	6	F	V	f	v	▌	▪	μ_4	÷	ф	ж	Ф	Ж
7	'	7	G	W	g	w	▟	⌘	т	≈	г	ь	Г	В
8	(	8	H	X	h	x	▐	▲	Ξ	°	х	ь	Х	Ь
9	)	9	I	Y	i	y	▌	↕	Θ	·	и	ы	И	Ы
A	*	:	J	Z	j	z	▐	ω	Ω	·	й	э	Й	Э
B	+	;	K	[	k	{	▒	■	δ	√	к	ш	К	Ш
C	,	<	L	\	l	!	▒	▬	∞	n	л	э	Л	Э
D	-	=	M	]	m	}	▼	▌	ø	2	м	щ	М	Щ
E	.	>	N	^	n	~	▲	▌	∈	▪	н	ч	Н	Ч
F	/	?	O	_	o	▲	▶	▬	∩	⌘	о	ъ	О	

Шестнадцатиричный код символа, стоящего на пересечении столбца p и строки q в таблице 1, равен $(pq)_{16}$. Его десятичное значение равно $16p+q$.

ПРИМЕР 1

СИМВОЛ	ШЕСТНАДЦАТИРИЧНЫЙ КОД	ДЕСЯТИЧНЫЙ КОД
W	57	87
пробел (_)	20	32
=	3D	61
курсор	FF	255

Константы. В Бейсике используются константы четырех тип строковые (или символьные); числовые целые; числовые вещес ные одинарной точности; числовые вещественные двойной ности.

Строковые константы записываются в виде последовательности произвольных литер языка, начинающейся и заканчивающейся кавычками ". То, что находится между кавычек, называется значением строковой константы, а количество символов значения — ее длиной (см. пример 2). Длина не должна превосходить 255.

```
ПРИМЕРЫ 2.    (L - длина константы)
"КОНСТАНТА"          , L=9
"FLOATING-POINT"     , L=14
"253.175"            , L=7
"A__B___C___"        , L=11
```

Числовые целые константы C лежат в диапазоне:

$$-32768 \leqslant C \leqslant 32767.$$

Записывать их можно в десятичной, шестнадцатиричной, восьмиричной и двоичной системах счисления (формах). При этом в трех последних случаях перед значением целой константы требуется проставлять соответствующее основание системы счисления в виде сочетания символов: &H(16), &O(8) или &B(2).

```
ПРИМЕРЫ 3.  Константы
746       , -2163    - целые десятичные,
&H22B     , &HFF     - целые шестнадцатиричные
                       (допустимые цифры :
                       0,1,...9,A,B,C,D,E,F),
&O1777    , &O123    - целые восьмиричные
                       (допустимые цифры :
                       0,1,...7),
&B1010011, &B0101    - целые двоичные
                       (допустимые цифры: 0,1).
```

Числовые вещественные константы двойной точности записываются в десятичном виде в фиксированной или плавающей формах. Количество цифр для фиксированной формы числа или количество цифр мантиссы для плавающей формы числа не превосходит 14.

ПРИМЕРЫ 4. Формы констант

1) фиксированная форма (двойная точность)

```
157.8521
34.1
5
93.64478#
```

2) плавающая форма (двойная точность)

```
1.22D-2        (1.22*10^-2)
71.21231D12    (71.21231*10^12)
```

Здесь символы # и D используются как указатели двойной точности. При плавающей форме записи константы символ D служит еще и разделителем мантиссы и порядка.

Числовые вещественные константы одинарной точности записываются в десятичном виде в фиксированной или плавающей формах. Количество цифр для фиксированной формы числа или количество цифр мантиссы для плавающей формы числа не превосходит 6.

ПРИМЕРЫ 5

1) фиксированная форма (одинарная точность)

```
5.171!
6!
731.1371!
12.56!
```

2) плавающая форма (одинарная точность)

```
3.74E5         (3.74*10^5)
1E-3           (1*10^-3)
```

Здесь символы ! и E используются как указатели одинарной точности. Кроме того, при плавающей форме записи вещественной константы символ E служит еще и разделителем мантиссы и порядка. При отсутствии указателей точности вещественные константы в фиксированной форме при количестве знаков более 6 или с ненулевой дробной частью считаются двойной точности.

Переменные. Имя (идентификатор) переменной есть последовательность из произвольного количества латинских букв, арабских цифр и, возможно, некоторых специальных символов алфавита. Однако при выборе имен следует иметь в виду такие ограничения. Первый символ обязан быть буквой. Значащими являются лишь первые два символа. Служебные слова нельзя использовать в качестве имен.

Тип переменной определяется типом принимаемых ею значений и может или задаваться последним символом в самом имени, или определяться командой DEF декларации типа. Остановимся на этих случаях подробнее.

1) Тип переменной указывается в имени последним знаком вида:

$ — для строковых переменных,
% — для целочисленных переменных,
# — для вещественных (действительных) переменных двойной точности,
! — для вещественных (действительных) переменных одинарной точности.

По умолчанию, т. е. при отсутствии в имени знаков: $, %, # и !, тип переменной считается вещественным с двойной точностью.

ПРИМЕРЫ 6

```
A!, B%, CDF#, MAP, L$,
E(I), F(4,R), E%(K%), E%(K), K$(L)
```

Здесь первые пять примеров определяют имена простых переменных, а последние пять — имена элементов массивов, в которых за простым именем в круглых скобках указываются один, два или более индексов.

2) Тип объявляется командой DEF:

DEFSTR — для строковых переменных,
DEFINT — для целочисленных переменных,
DEFDBL — для вещественных переменных двойной точности,
DEFSGN — для вещественных переменных одинарной точности.

ПРИМЕРЫ 7

```
DEFSTR M, L, RES  - все переменные, начина-
                    ющиеся с букв M, L и R,
                    об'являются строковыми;
DEFINT A, K-Z     - все переменные, начина-
                    ющиеся с буквы A и с
                    букв диапазона от K до
                    Z об'являются целочис-
                    ленными;
DEFDBL E-G        - все переменные, начина-
                    ющиеся с букв E, F и G,
                    об'являются веществен-
                    ными с двойной точно-
                    стью;
```

```
DEFSGN E,GAME      - все переменные, начинающиеся с букв E и G,
                     об'являются вещественными с одинарной точностью.
```

Заметим, что объявление типа командой DEF распространяется и на массивы, имена которых начинаются с соответствующих букв, но не действует на переменные и массивы с явным указанием типа знаками: $, %, # и !.

Выражения. Выражения могут быть константами, переменными или комбинациями констант, переменных и функций, соединенных знаками операций так, что в результате выполнения соответствующих действий получается единственное значение. Все символы выражения располагаются последовательно друг за другом. Никаких верхних или нижних индексов в записи не допускается. Для указания приоритетности при вычислениях можно использовать круглые скобки. Различают 4 типа операций. Рассмотрим их.

Арифметические операции. В таблице 2 в порядке уменьшения приоритета перечислены 8 арифметических операций. В качестве разделителя при перечислении возможных вариантов там используется символ ▌.

Таблица 2

	Знак операции	Название	Примеры выражений
1	^	возведение в степень	X^Y
2	—	вычитание (одноместное)	—U
3—4	* ▌ /	умножение, деление	X*Y X/Y
5—6	+ ▌ —	сложение, вычитание (двуместное)	X+Y X—Y
7	\	целочисленное деление	X\Y 10\4 (2) 25.1\6.9 (4)
8	MOD	нахождение остатка	X MOD Y 10 MOD 3 (1)

Если в выражении присутствуют лишь числовые константы, переменные, функции и только знаки арифметических операций, то такое выражение будем называть арифметическим.

ПРИМЕРЫ 8

АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ	АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ НА БЕЙСИКЕ
$ab^2 + \frac{c+d}{e}$	A*B^2+(C+D)/E
$\frac{x\sin x}{y} + 3 \over 1 + \ln z$	(X*SIN(X)/Y+3)/(1+LOG(Z))

Строковые операции. Строковыми (символьными) выражениями являются строковые константы, переменные и функции. О последних речь пойдет в пункте 1.2. Выполняя над строковыми выражениями операцию конкатенации (+), или, по-другому, присоединения, мы снова получаем строковое выражение. Смысл операции конкатенации разъясняет приведенный ниже пример.

ПРИМЕР 9

```
10 A$="КОНТРА":B$="БАС"          run
20 PRINT A$+B$                   КОНТРАБАС
30 PRINT "НОВЫЙ "+A$+B$          НОВЫЙ КОНТРАБАС
```

Операции отношения. Простым выражением отношения называют два арифметических или два символьных выражения, разделенных знаком операции отношения.

Таблица 3

Операции отношения

Знак операции		Название	Знак операции		Название
1	=	равно	4	>	больше
2	<>	не равно	5	<=	меньше или равно
3	<	меньше	6	>=	больше или равно

ПРИМЕРЫ 10

```
X>3, R<Z(3,4)        B$="ПОВЕРХНОСТЬ"
A^2+B^2<=C^2
```

Значениями простого выражения отношения являются элементы множества {T, F} (TRUE — истина, FALSE — ложь). При этом если соответствующее отношение удовлетворяется, то его значением бу-

дет T, в противном случае — F. Используются выражения отношения в операторах IF (если) в качестве проверяемых условий для организации разветвлений при вычислениях.

Значения двух символьных выражений сравниваются посимвольно слева направо. При этом учитывается, что литеры алфавита упорядочены в соответствии с их шестнадцатиричными кодами и в возрастающем порядке располагаются так (см. таблицу 1):

пробел ! " # ... Щ Ч курсор

Логические операции. Используя простые выражения отношения с помощью логических операций (связок), получают выражение отношения более общего вида. Допустимы следующие связки: NOT (нет), AND (и), OR (или), XOR (исключающее или), EQV (эквиваленция) и IMP (импликация). В таблице 4 приведены значения истинности для подобных «составных» выражений отношения.

Таблица 4

X	Y	NOTX	XANDY	XORY	XXORY	XEQVY	XIMPY
T	T	F	T	T	F	T	T
T	F	F	F	T	T	F	F
F	T	T	F	T	F	F	T
F	F	T	F	F	F	T	T

ПРИМЕРЫ 11

```
(X>Y) AND (C=D)     (A^2+B^2<=C^2) OR (A EQV B)
```

Заметим, что логические операции можно выполнять над значениями любых выражений *x* и *y*. Реализуются эти операции над соответствующими битами данных по правилам таблицы 4, где символ T везде заменен на 1, а символ F — на 0.

1.2. Функции

В Бейсике различают 4 основных типа функций: числовые функции; функции для работы со строками; функции преобразования; функции пользователя.

Первые три из них относятся к так называемым *стандартным функциям*, вычисление значений которых реализуется специальными программами, находящимися в памяти ЭВМ. Каждая такая программа имеет уникальное имя (SIN, EXP, MID$, ASC и т. п.), по которому и происходит обращение к ней. К четвертому типу относятся те функции, которые пользователь определяет в программе сам.

Остановимся на каждом типе функций в отдельности.

Числовые функции. Аргументом любой стандартной числовой функции может служить произвольное арифметическое выражение, заключенное в круглые скобки.

	Функция	Название	Выполняемые действия
1	ABS (X)	Абсолютная величина	Вычисляется величина $\|x\|$
2	LOG (X)	Натуральный логарифм	$\ln x$
3	EXP (X)	Экспонента	e^x
4	SQR (X)	Квадратный корень	Square root — квадратный корень Вычисляется величина $\sqrt{x}$
5	SIN (X)	Синус	$\sin x$ (x в радианах)
6	COS (X)	Косинус	$\cos x$ (x в радианах)
7	TAN (X)	Тангенс	$\mathrm{tg}\, x$ (x в радианах)
8	ATN (X)	Арктангенс	$\mathrm{arctg}\, x$
9	SGN (X)	Знак числа	Signum — знак. Вычисляется значение функции: $\mathrm{sign}\, x = \begin{cases} 0, & \text{если } x=0, \\ 1, & \text{если } x>0, \\ -1, & \text{если } x<0 \end{cases}$
10	INT (X)	Целая часть	Integer — целый. Находится максимальное целое число, меньшее или равное x
11	FIX (X)	Фиксация (отбрасывание дробной части)	Fix — фиксация. У значения x отбрасываются десятичная точка и все цифры, следующие за ней
12	RND (X)	Генерирование случайных чисел	Random случайный. Генерируется псевдослучайное число из диапазона]0, 1[
13	CINT (X)	Преобразование к целому типу	Convert to integer type. X преобразуется к целому типу
14	CDBL(X)	Преобразование к двойной точности	Convert to double precision type. X преобразуется к типу двойной точности
15	CSNG (X)	Преобразование к одинарной точности	Convert to single precision type. X преобразуется к типу одинарной точности

Остановимся на двух моментах.

Во-первых, при вычислении значений тригонометрических функций SIN(X), COS(X) и TAN(X) аргумент X необходимо выразить в радианах. И здесь полезно помнить, что $4 \times \mathrm{ATN}(1) = \pi$.

Во-вторых, более подробно опишем функцию RND(X), используемую во многих игровых и моделирующих программах.

Аргумент X (X>0) для RND (X) является псевдопеременной и не влияет на значение функции. Обычно берут X=1. При запуске конкретной программы отдельные вычисления RND(1) приводят к выдаче равномерно распределенной на интервале (0,1) числовой последовательности следующего вида:

```
. 59521943994623
. 10658628050158
. 76597651772823
. 57756392935958
 ...  ...  ...
```

Повторные запуски этой программы генерируют ту же самую последовательность. Этот факт оказывается удобным при отладке программ, т. е. при поиске в них тех или иных ошибок. Чтобы добиться появления другой последовательности, необходимо до использования функции RND (X) произвести так называемую инициализацию генератора случайных чисел любым неположительным числом. Для этого достаточно выполнить команду Y=RND (X), где X≤0. По умолчанию подобная инициализация выполняется с X=0 при каждом запуске программы. Если взять X=—TIME, т. е. реализовать команду Y=RND (—TIME), то затем от запуска к запуску будем получать непредсказуемые меняющиеся равномерно распределенные на (0,1) последовательности случайных чисел.

ПРИМЕРЫ 12

```
1. INT(10*RND(1))      - генерирование деся-
                         тичной цифры от 0
                         до 10.
2. A+(B-A)*RND(1)      - генерирование слу-
                         чайного числа из
                         интервала (A,B).
3. 10 X=RND(-TIME)     - получение непред-
   20 FOR X=1 TO 500     сказуемой равномер-
   30 PRINT RND(1)       но распределенной
   40 NEXT X             на (0,1) последо-
                         тельности из 500
                         чисел.
```

Строковые функции. Здесь будут рассмотрены следующие 5 функций:

1. MID$ — функция (middle — середина),
2. LEFT$ — функция (left — левый),
3. RIGHT$ — функция (right — правый),
4. STRING$— функция (string — строка),
5. SPACE$ — функция (space — пространство).

Другие функции для работы со строками рассматриваются в пункте «Функции преобразования».

Функции 1—3 используются справа и слева от знака равенства в командах LET (пусть) и условной передачи управления IF (если). Функции 4 и 5 используются для инициализации новых строк.

MID$-функция. Общий вид MID$-функции следующий: MID$ (α, m [,n]), где: α — строковое выражение, m, n — арифметические выражения.

Целые части значений выражений m и n будем обозначать соответственно через M и N. По умолчанию N равно количеству символов значения α от позиции M и до конца строки.

Рассмотрим возможные случаи.

а) Функция MID$ слева от знака равенства в команде LET.

$$\text{LET MID\$ } (\alpha,\ m\ [,n]) = \beta \qquad (1)$$

Пусть L — длина значения символьного выражения β. При выполнении команды LET последовательные символы α, начиная с позиции M, замещаются на первые min (L, N) символов строки β. Остальные символы в α остаются без изменения.

Заметим, что в рассматриваемом случае α должно быть определено до выполнения команды (1).

ПРИМЕР 13

```
10 X$="1234567890"          run
20 INPUT Y$,M,N             ? LL,5,3
30 MID$(X$,M,N)=Y$          1234LL7890
40 PRINT X$                 run
                            ? ПЕТЯ,4,3
                            123ПЕТ7890
                            run
                            ? КРОКОДИЛ,7,4
                            123456КРОК
```

б) Функция MID$ справа от знака равенства в команде LET.

$$\text{LET}\gamma = \text{MID\$ } (\alpha,\ m\ [,n]) \qquad (2)$$

При выполнении команды LET N последовательных символов значения α, начиная с позиции M, становятся значением переменной γ. Предварительное определение γ не обязательно. При выполнении команды (2) возможно увеличение длины значения γ.

ПРИМЕР 14

```
1 Z$="123"                    run
2 INPUT X$,M                  ? JCUKENGEJZH,5
3 Z$=MID$(X$,M)               ENGEJZH
4 Y$=MID$(X$,M)               ENGEJZH
5 PRINT Z$
6 PRINT Y$
```

в) Функция MID$ в команде IF.

$$\text{IF } \gamma * \text{MID\$}(\alpha, \text{m } [,\text{n}]) \text{ THEN P} \qquad (3)$$

где: $*\in\{<, >, =, \leqslant, \geqslant, <>\}$; P — номер программной строки; γ — символьное выражение.

При выполнении команды (3) прежде всего из α, начиная с позиции M, выбирается N последовательных символов. Получившаяся строка β сравнивается с γ. Если выражение отношения $\gamma * \beta$ имеет значение T (истинно), то управление передается первой команде строки P. В противном случае начинает выполняться команда, следующая за IF.

LEFT$-функция. LEFT$-функция является частным случаем MID $-функции и через нее может быть определена так:

$$\text{LEFT } \$(\alpha, \text{n}) = \text{MID\$}(\alpha, 1, n)$$

Иными словами, значением LEFT $ (α, n) является строка из N последовательных левых символов значения строкового выражения α.

ПРИМЕР 15

```
10 INPUT X$,N                 run
20 U$=MID$(X$,1,N)            ? FYWAPROLDV,6
30 V$=LEFT$(X$,N)             FYWAPR
40 PRINT U$                   FYWAPR
50 PRINT V$
```

RIGHT$-функция. RIGHT$-функция является частным случаем MID $-функции и через нее может быть определена так:

$$\text{RIGHT } \$(\alpha, \text{n}) = \text{MID\$}(\alpha, \text{L} - n + 1, n)$$

где L — длина значения α. Иными словами, значением RIGHT $ (α, n) является строка из N последовательных правых символов значения строкового выражения α.

ПРИМЕР 16

```
10 INPUT X$,N                 run
20 U$=MID$(X$,LEN(X$)-N+1,N)  ? 1234567890,4
30 V$=RIGHT$(X$,N)            7890
40 PRINT U$                   7890
50 PRINT V$
```

STRING$-функция. Общий вид STRING$-функции следующий: STRING (n, α), где: α — строковое выражение; n — арифметическое выражение.

Значением этой функции является последовательность из N==INT (n) одинаковых символов, равных первой литере значения α. С помощью STRING$-функции можно назначать длину и проводить начальную инициализацию строковых переменных. В STRING$-функции α может быть не только строковым выражением, но и кодом ASCII. Тогда ее значением будет последовательность из N символов алфавита или управляющих сигналов с данным кодом α.

ПРИМЕР 17

```
10 Y$=STRING$(10,"*")         run
20 PRINT LEN(Y$)               10
30 PRINT "#"+Y$+"FILE"        #**********FILE
```

SPACE $-функция. SPACE$-функция является частным случаем STRING $-функции и через нее может быть определена так:

$$\text{SPACE\$ } (\text{n}) = \text{STRING\$ } (\text{n}, "\sqcup")$$

С помощью SPACE$-функции назначается длина и проводится начальная инициализация строковых переменных пробелами.

Функция преобразования. Речь пойдет о следующих функциях.

1. LEN (α) — длина значения строкового выражения α.
2. INSTR — номер позиции вхождения одного строкового значения в другое.
3. HEX$ (β) — функции преобразования десятичных чисел β OCT$ (β) в строковую форму с представлением их в BIN$ (β) шестнадцатиричном, восьмиричном, двоич- STR$ (β) ном или десятичном изображениях.
4. VAL (α) — десятичное число, получаемое преобразованием символьного представления α к числовому типу.

5. ASC (α) — десятичный код первого символа значения строкового выражения α.
6. CHR$ (β) — символ с десятичным кодом β.

Остановимся на этих функциях подробно.

LEN-функция. Значение функции LEN (α) равно длине значения строкового выражения α.

ПРИМЕР 18

```
10 X$="AB__CD__E___"                    run
20 S=LEN(X$)                             12
30 PRINT S
```

INSTR-функция. Функция INSTR ([n], α, β), где: α, β — строковые выражения; n — арифметическое выражение, равна номеру позиции в значении α, считая от его начала, с которой значение β входит от позиции N=INT (n) и дальше в α. По умолчанию n=1.

ПРИМЕР 19

```
10 INPUT X$,Y$,U          run
20 L=INSTR(U,X$,Y$)       ? ABCRSTRSTRSTP,RST,1
30 PRINT L                 4
                          run
                          ? ABCRSTRSTRSTP,RST,8
                           10
```

HEX$, OCT$, BIN$ и STR$-функции. Значением функций HEX$ (γ), OCT$ (γ), BIN$ (γ) и STR$ (γ) являются строковые формы соответственно шестнадцатиричного, восьмиричного, двоичного или десятичного представления десятичного числа γ.

ПРИМЕР 20

```
10 INPUT X
20 U1$=HEX$(X):U2$=OCT$(X)
30 U3$=BIN$(X):U4$=STR$(X)
40 PRINT U1$+" "+U2$+" "+U3$+" "+U4$
run                                    run
? 123                                  ? 444
7B 173 1111011  123                    1BC 674 110111100  444
```


VAL-функция. Функция VAL (α), где α — строковое выражение, имеет значением десятичное число, полученное преобразованием его символьного представления в α к числовому типу.

ПРИМЕР 21

```
10 INPUT Z$                              run
20 X=VAL(Z$)+1                           ? 499
30 PRINT X                               500
```

ASC-функция. Функция ASC (α), где α — строковое выражение, имеет значением десятичный код первого символа значения α.

ПРИМЕР 22

```
10 INPUT X$:Y=ASC(X$)
20 PRINT X$,Y
run                run
? J                ? αδβ
J      187         αδβ      165    (код α)
```

CHR$-функция. Функция CHR$ (X) имеет значением символ, десятичный код которого равен X.

ПРИМЕР 23

```
10 INPUT X:Y$=CHR$(X)          run
20 PRINT X,Y$                  ? 164
                                164          Σ
                               run
                               ? 187
                                187          J
```

Имеет смысл обратить внимание на то, что в области определения функций ASC и CHR$ справедливы соотношения:

ASC (CHR$ (β)) = β (β — десятичное число), CHR$ (ASC (α)) = α (α — символ).

Другими словами, функция ASC обратна к CHR$ и, наоборот, функция CHR$ обратна к ASC.

Функции пользователя. Пользователь имеет возможность определить в программах новые функции. Они должны быть описаны командой DEF (*definition* — определение) следующим образом:

$$\text{DEFFN}\,\alpha\,[(\beta_1[,\beta_2]\ldots)] = \gamma \qquad (4)$$

где:

FN α — имя новой функции (FN — его фиксированная часть, а α выбирается по усмотрению пользователя);
$\beta_1, \beta_2 \ldots$ — имена переменных величин (аргументы функции);
γ — выражение, зависящее от $\beta_1, \beta_2, \ldots$

Если описание команды (4) дано, то затем функцию

$$\text{FN}\,\alpha\,[(\sigma_1[,\sigma_2]\ldots)] \qquad (5)$$

где σ_k — выражения, имеющие тот же тип, что и β_k ($k=1,2,\ldots$), можно использовать в любых выражениях, входящих в те или иные команды программы. Значение FN α вычисляется так. В γ каждому β_k присваивается значение σ_k ($k=1,2,\ldots$). Затем вычисляется γ и результат присваивается величине (5).

Величины β_k в (4) и σ_k в (5) ($k=1,2,\ldots$) называются соответственно формальными и фактическими параметрами.

ПРИМЕРЫ 24

```
1) 10 DEFFNP(X)=X^2+X+1
   20 PRINT FNP(1);FNP(10);FNP(2)^2
   30 INPUT A
   40 PRINT FNP(A)
   run
    3  111  49
   ? 5
    31
```

```
2) 10 DEFFNZ(A,B)=A^2+B^2
   20 INPUT X,Y
   30 PRINT FNZ(X,Y);FNZ(Y,8)
```

```
run
? 3,4
 25  80
```

```
3) 10 DEFFNS$(X$,K)=MID$(X$,K,1)
   20 INPUT X$,K
   30 PRINT FNS$(X$,K)
```

```
run
? TABLE,4
L
```

1.3. Дисплей и клавиатура

Клавиатура является устройством ручного ввода программ и данных, кодируемых последовательными нажатиями расположенных на ней клавиш.

Дисплей, наряду с клавиатурой, является основным устройством оперативного взаимодействия человека и ЭВМ. Вводимые с клавиатуры символы визуально отображаются на экране дисплея и легко могут быть отредактированы. Получаемая в результате вычислений информация также может выводиться на экран в виде текстов, рисунков, графиков, диаграмм и т. п. Дисплеи бывают двух типов: цветные и монохромные. Отдельные участки экрана цветного дисплея и изображения на них допускают 16 различных цветов (оттенков). На монохромных дисплеях возможно до 16 оттенков одного, обычно зеленого, цвета.

Экраны дисплея. Физически дисплей имеет 1 экран. Однако качественных состояний (режимов), в которых он может находиться,— 4. Эти состояния пронумеруем от 0 до 3 и далее, всякий раз речь будем вести о конкретном экране K (K=0, 1, 2, 3).

Экраны 0 и 1 являются символьными. На них можно индицировать только символы алфавита Бейсика и ряд дополнительных символов псевдографики. Экраны 2 и 3 являются графическими. На них возможен вывод и графической, и текстовой информации. Переключение с одного экрана на другой осуществляется командой SCREEN (экран). При включении компьютера автоматически устанавливается экран 0.

О графических экранах речь пойдет в пункте 2.1. Сейчас же сосредоточим свое внимание на символьных экранах, где можно выделить две основные зоны: бордюр и фон. На экране 0 эти зоны всегда одного цвета. На экране 1 их цвета могут быть различными. Схематическое изображение символьных экранов дано на рисунке 1. Программы и данные в виде последовательности символов алфавита Бейсика отображаются на фоне экрана. Фон условно разделен на 24 строки, которые пронумерованы сверху вниз числами 0, 1, ..., 23. Последняя строка называется строкой-подсказкой. На ней индицируются начальные символы значений спецклавиш (ключей). Пусть k_1 — последняя строка экрана, на которой могут отображаться символы, вводимые с клавиатуры. Обычно $k_1=22$. Однако можно сделать $k_1=23$, превратив строку-подсказку в нормальную строку экрана.

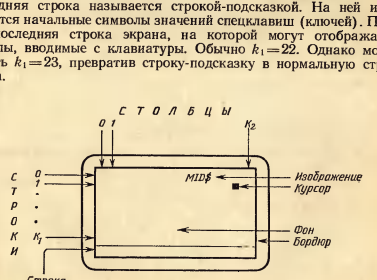


Рис. 1

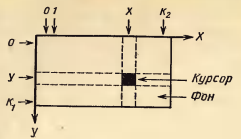


Рис. 2

Условно по вертикали разделим фон на столбцы и пронумеруем их слева направо последовательными числами 0, 1, ..., k_2. Тогда по умолчанию для экрана 0 k_2=38 (39 столбцов), а для экрана 1 k_2=28 (29 столбцов). В общем случае k_2 можно изменять по команде WIDTH в пределах: $0 \leqslant k_2 \leqslant 39$ (для экрана 0), $0 \leqslant k_2 \leqslant 31$ (для экрана 1).

Параметры k_1 и k_2 определяют поле ввода (фон) из $(k_1+1)\times(k_2+1)$ позиций (x, y), в каждой из которых может быть проиндицирован ровно один символ. Здесь x — номер столбца ($0 \leqslant x \leqslant k_2$), а y — номер строки ($0 \leqslant y \leqslant k_1$). При вводе данных с клавиатуры в любой конкретный момент времени одна из позиций поля ввода отмечена специальным знаком ■, называемым *курсором*, или *маркером*. Пару (x, y) можно считать координатами курсора (см. рис. 2). Пусть позиции фона упорядочены следующим образом:

(0, 0), (1, 0), ..., (k_2, 0), (0, 1), ..., (k_2, 1), ..., (k_2, k_1)

Переход курсора от (x, y) к последующей позиции назовем его движением «вперед», а к предыдущей — движением «назад». При этом последующей для ($k_2 k_1$) и предыдущей для (0, 0) считаем сами эти позиции. Символ, набираемый на клавиатуре, появляется в текущей позиции курсора. При этом сам курсор смещается на одно знакоместо «вперед». Иными словами, при вводе символов курсор последовательно, каждый раз на 1 позицию, сдвигается вправо по текущей строке экрана, пока не дойдет до ее края. Затем он переходит на начало следующей строки и т. д. Исключением является позиция ($k_2 k_1$), при которой набор очередного символа приводит к дополнительному перемещению всего изображения по фону на 1 строку вверх. Причем бывшая нулевая строка исчезает, а для ввода предоставляется строка k_1 с курсором на месте (0, k_1).

Управляющие клавиши. На клавиатуре имеется ряд так называемых управляющих клавиш, которые служат не для ввода символов, а для задания Бейсик-системе тех или иных команд, связанных с очисткой экрана, стиранием символов, перемещением курсора и т. п. В таблице 6 описывается назначение этих клавиш. При этом если некоторое действие производится при одновременном нажатии двух и более клавиш, то между их наименованиями ставится символ &.

Назначение управляющих клавиш

N	Управляющие клавиши	Название Выполняемые действия
1	⇒ CTRL & \	Курсор «вперед». Курсор сдвигается на 1 позицию «вперед»
2	⇐ CTRL &]	Курсор «назад». Курсор сдвигается на 1 позицию «назад»
3	⇑ SHIFT & CTRL & ^	Курсор вверх. Курсор сдвигается, если это возможно на 1 позицию вверх
4	⇓ SHIFT & CTRL & _	Курсор вниз. Курсор сдвигается, если это возможно, на 1 позицию вниз
5	CTRL & B	Курсор в начало слова
6	CTRL & F	Курсор на следующее слово
7	CTRL & N	Курсор в конец строки
8	HOME CTRL & K	Домой. Курсор перемещается в левый верхний угол экрана (фона)
9	DEL	Стирание символа в позиции курсора. Все символы логической строки от позиции за курсором и до ее конца сдвигаются на одну позицию «назад». Символ в позиции курсора пропадает, а в последней позиции строки появляется пробел
10	BS CTRL & H	Стирание символа слева от курсора. Все символы логической строки от позиции курсора и до ее конца сдвигаются на одну позицию «назад». Символ, находящийся перед курсором, пропадает, а в последней позиции строки появляется пробел
11	CTRL & E	Стирание с экрана конца строки. Все символы строки от позиции курсора и до ее конца замещаются пробелами
12	CTRL & U	Стирание с экрана текущей строки. Все символы строки, отмеченной курсором, замещаются пробелами
13	CLS CTRL & L	Очистка экрана. CLS (clear *screen* — очистка экрана). Изображение стирается с экрана
14	↵ CTRL & M	Ввод текущей строки. Программная текущая строка фиксируется в памяти. Строка непосредственного счета выполняется
15	INS CTRL & R	Переключение на режим вставки. Задание или отмена режима вставки, при котором курсор уменьшается в два раза и набор любого символа α приводит к сдвигу части логической строки от курсора и дальше на одну позицию «вперед». На месте курсора появляется α

Продолжение

N	Управляющие клавиши	Название. Выполняемые действия
16	TAB CTRL & I	Табуляция. Курсор смещается в ближайшую правую позицию текущей строки, кратную 8. Если такой позиции нет, то он переходит в начало следующей строки. Проходимые курсором места в любом случае замещаются пробелами
17	STOP	Приостанов-пуск. При вычислениях по программе нажатие на эту клавишу приводит к их приостанову. Повторное нажатие на STOP возобновляет вычисления с прерванного места
18	CTRL & STOP	Прерывание. Прерывание вычислений. При счете по программе на экран выдается сообщение о номере строки, на которой произошло прерывание
19	РУС	Переключатель русского шрифта. Задание или отмена режима ввода русской символики. При включении режима РУС на соответствующей клавише загорается индикаторная лампочка
20	CAPS	Переключатель заглавных и строчных букв. Задание или отмена режима ввода заглавных букв. При включении режима CAPS на соответствующей клавише загорается индикаторная лампочка
21	GRAPH SHIFT & GRAPH	Регистры дополнительных значений клавиш (псевдографика). Каждая клавиша α определяет два дополнительных значения кроме тех, которые обозначены на ней. Для получения этих значений необходимо одновременное нажатие на клавиши: GRAPH и α или SHIFT, GRAPH и α
22	F1/F6 F2/F7 F3/F8 F4/F9 F5/F10	Функциональные клавиши. Имеется 10 функциональных клавиш (ключей). По умолчанию они имеют такие значения: F 1 — print F 2 — auto F 3 — goto F 4 — list F 5 — run F 6 — run "COM:" ↵ F 7 — save "COM:" ↵ F 8 — load "COM:" ↵ F 9 — merge "COM:" ↵ F10 — comterm Первые 5 значений ключей или их начальные части инициируются в 23 строке — подсказке. Остальные появляются там при нажатии клавиши SHIFT. Ввод любого из этих значений организуется одним нажатием на соответствующую клавишу (возможно вместе с SHIFT). Значения спецклавиш можно менять, а их засветку из 23 строки — убрать.

Команды для работы с экранами. В этом пункте речь пойдет о командах: SCREEN, COLOR, CLS, WIDTH, KEY OFF, KEY ON, KEY LIST и KEY, применяемых и в непосредственном счете, и при вычислениях по программе. С их помощью можно выбрать тип экрана и производить его чистку, устанавливать цвет бордюра, фона и изображения, задавать ширину фона, удалять надписи со строки-подсказки и восстанавливать их снова, индицировать, а также изменять значения функциональных клавиш. В некоторых из указанных команд могут присутствовать по несколько параметров. Неиспользуемые параметры можно не указывать, но если они не последние в записи, то ограничивающие их запятые обязательны. При этом каждый назначенный параметр действует до тех пор, пока не получит иное значение в соответствующей последующей команде.

SCREEN. Общая форма записи команды SCREEN (экран) будет дана в пункте 2.1. Сейчас же ограничимся частным случаем.

SCREEN n (1)

где $n \in \{0, 1, 2, 3\}$. По команде (1) назначается экран номер n. По умолчанию n=0.

n=0. Символьный экран, имеющий 24 строки и максимум 40 столбцов. Цвета бордюра и фона всегда совпадают.

n=1. Символьный экран, имеющий 24 строки и максимум 32 столбца. Цвета бордюра и фона могут быть различными.

n=2, 3. Графические экраны (см. 2.1).

Заметим, что команду INPUT можно использовать только при символьных экранах. Далее, команды графики допустимы лишь при графических экранах.

COLOR. Записываемая команда COLOR (цвет) в виде COLOR [α] [,β] [,γ], где α, β и γ — арифметические выражения, целые части значений которых являются кодами цветов изображения (*a*), фона (*b*) и бордюра (*c*). Они должны лежать в пределах от 0 до 15. В таблице 7 дано соответствие цветов и кодов для одного из типов многоцветных дисплеев. Для монохроматических дисплеев эти коды определяют различные оттенки одного и того же цвета.

Таблица 7

Код	Цвет	Код	Цвет
0	прозрачный	8	красный
1	черный	9	светло-красный
2	зеленый	10	темно-желтый
3	светло-зеленый	11	светло-желтый
4	темно-синий	12	темно-зеленый
5	синий	13	сиреневый
6	темно-красный	14	серый
7	голубой	15	белый

Каждый параметр в COLOR действует до нового назначения. Любой из них можно опустить. Однако если он не последний в записи, то необходимо обозначить его запятой. Значение пропущенного параметра и соответствующего цвета не изменяется.

Использование команды COLOR позволяет выводить на экран разноцветные тексты и рисунки, подбирать для них цвет фона, а для экранов 1, 2 и 3 и цвет бордюра.

П Р И М Е Р Ы 25

```
1)  COLOR 1,7,3
    COLOR ,11
    COLOR X,,Z

2) Удобные для работы сочетания цветов
      Зеленый                Цветной
      дисплей                дисплей
COLOR Ø,1,1Ø          COLOR Ø,1,1Ø
COLOR Ø,11,Ø          COLOR 1,7,13
COLOR 1,15,8          COLOR 1,11,12

3)  1Ø SCREEN 1
    2Ø I=INT(RND(1)*16)
    3Ø F=INT(RND(1)*16)
    4Ø B=INT(RND(1)*16)
    5Ø COLOR I,F,B
    6Ø LOCATE 9,1Ø: PRINT I;F;B
    7Ø FOR K=1 TO 7ØØ:NEXT K
    8Ø GOTO 2Ø
```

Остановимся на некоторых особенностях реализации команды COLOR. На символьных экранах цвет изображения при выполнении COLOR меняется сразу на всем экране, и потому многоцветные тексты здесь недопустимы. На графических экранах назначение нового цвета изображения не распространяется на ранее индицированные тексты или рисунки. Этим самым предоставляется возможность вывода многоцветных изображений. Далее, если на символьных экранах цвет фона меняется по одной команде COLOR, то на графических — фон можно сменить только парами команд COLOR и SCREEN или COLOR и CLS. При этом, разумеется, изображение будет потеряно.

Заметим, что прозрачный цвет есть текущий цвет бордюра.

CLS. По команде CLS (*clear screen* — очистка экрана) стирается изображение на символьных и графических экранах.

Точнее, цвет изображения сливается с цветом фона.

П Р И М Е Р 26

```
1ØØ INPUT A
11Ø IF A=Ø THEN CLS
```

WIDTH. Команда WIDTH (ширина) используется для установки на символьных экранах условных вертикальных границ по центру фона. В полосе между этими границами и будет фиксироваться изображение. Записывается команда в виде WIDTH α, где α — арифметическое выражение, целая часть значения которого должна лежать в диапазоне от 1 до 40 (для экрана 0) и от 1 до 32 (для экрана 1). По умолчанию для экрана 0 $\alpha=39$, а для экрана 1 $\alpha=29$. При выполнении команды WIDTH имеющееся на фоне изображение исчезает, а последующая индикация символов организуется в соответствии с новым назначением «ширины».

П Р И М Е Р Ы 27

```
1) 1Ø SCREEN Ø:WIDTH 25
2) 1ØØ SCREEN 1:WIDTH 28
   11Ø PRINT "ПРОВЕРКА"
   12Ø FOR K=1 TO 1ØØØ:NEXT
   13Ø WIDTH 1
```

Для изменения ширины экрана без его чистки необходимо использовать команду POKE (см. п. 1.5.).

KEY OFF. По команде KEY OFF (убрать ключ) из 23 строки-подсказки символьных экранов удаляется засветка значений ключей F1 — F10. Тем самым эта строка становится доступной для ввода информации. При этом сами значения ключей не разрушаются и по-прежнему могут использоваться простым нажатием на соответствующую функциональную клавишу F1—F10 (см. табл. 6).

П Р И М Е Р Ы 28

```
1) KEY OFF
2) 1Ø KEY OFF:LOCATE 5,23
   2Ø PRINT "ПРОВЕРКА KEY OFF"
```

KEY ON. Команда KEY ON (восстановить ключ) отменяет действие команды KEY OFF и индицирует на 23-й строке экрана те-

кущие значения ключей F1 — F10. Тем самым эта строка становится недоступной для ввода.

KEY LIST. По команде KEY LIST (список ключей) на экран дисплея построчно выводятся полные текущие значения функциональных клавиш F1 — F10. Это бывает необходимо в связи с тем, что любое конкретное значение ключа может содержать до 15 символов и не всегда целиком размещается на отводимом для него месте в строке-подсказке.

KEY. Для изменения значений функциональных ключей предназначена команда

KEY α, t (2)

где α — арифметическое, а t — строковое выражения. При этом число n=int (α) определяет номер функциональной клавиши и, следовательно, должно находиться в пределах от 1 до 10.

При выполнении команды (2) значение клавиши Fn становится равным значению t. Длина последнего не должна превышать 15. В противном случае будут использованы лишь первые 15 символов значения t. Заметим, что в ключе с помощью CHR$-функции можно задавать коды непечатаемых символов.

ПРИМЕРЫ 29

```
1) KEY 5,"print X;Y"

2) KEY 2,""    (Чистка значения ключа 2)

3) KEY 4,"LIST"+CHR$(13)

4) 10 K$=CHR$(34)
   20 KEY 5,"LOAD"+K$+"ABC"+K$+CHR$(13)
```

В примере 29(3) инициализация ключа приводит к тому, что нажатие на клавишу F4 будет вызывать индикацию текста программы без дополнительной команды «ввод строки» (←). Пример 29(4) демонстрирует задание знака кавычки в строковом значении.

Управление курсором. Оформлению результатов вычислений на экране дисплея и в некоторых иных случаях помогают многочисленные средства управления курсором. К ним следует отнести команду позиционирования LOCATE, функции CSRLIN, POS, TAB и SPC, а также серию CHR$-кодов (см. таблицу 8). Параметры в LOCATE, TAB и SPC могут быть произвольными арифметическими выражениями, у которых используются целые части значений. Функции TAB и SPC, а также все CHR$-коды применяются только в команде PRINT.

	Команда, функция. Код. Примеры	Выполняемые действия
1	LOCATE X, Y [, α] Команда позиционирования курсора по экрану Примеры 10 LOCATE 25,8 20 LOCATE 10,7 30 LOCATE A+B, C	LOCATE — размещать. Курсор перемещается из текущей позиции в позицию (X, Y), где X — номер столбца (X≥0), Y — номер строки (Y≥0). Величина α является переключателем типа курсора и может принимать значения 0 и 1: где: 0 — курсор отображается на экране только при некоторых командах ввода данных, 1 — курсор отображается По умолчанию α=0
2	CSRLIN Номер строки Пример 10 Y=CSRLIN 20 IFY = 23 THEN 50	Значением функции является номер строки курсора (Y — координата)
3	POS (0) Номер столбца Пример 10 Z=POS (0) 20 F=24 * CSRLIN + POS (0)	Значением функции является номер столбца курсора (X — координата)
4	TAB (n) Курсор в позицию N (N=int (n)) Пример 10 P=5 20 PRINTA; TAB (P); B; TAB (2*P); C	Функция TAB (n) используется в команде PRINT. Пусть T — текущая позиция курсора в строке. Возможны случаи. 1) T ≤ N. TAB перемещает курсор на N позиций «вперед», считая от позиции 0 текущей строки экрана. Одновременно производится заполнение пробелами всех позиций от T до N включительно. 2) T > N. TAB никаких действий не производит
5	SPC (n) Индикация N пробелов (N=int (n)) Пример 70 PRINTU; SPC (4); V	Функция SPC (n) используется в команде PRINT. По этой функции, начиная с текущей позиции курсора, индицируется N пробелов
6	CHR $ (8) Влево Пример 5 PRINT "ПАРА"; CHR $ (8); "ОХОД" (ПАРОХОД)	Управляющий код, по которому осуществляется возврат курсора на 1 позицию назад

Продолжение

	Команда, функция. Код. Примеры	Выполняемые действия
7	CHR $ (9) На метку табуляции Пример 7 PRINT 3; CHR $ (9); "B" (3 B)	Управляющий код, по которому курсор перемещается на следующую метку табуляции (Ø, 8, ...)
8	CHR $ (1Ø) На следующую строку Пример 1Ø PRINT "ЛЕВ" CHR $ (10); "ТИГР" (ЛЕВ ТИГР)	Управляющий код, по которому курсор перемещается на 1 строку вниз. При необходимости производится подтяжка строк
9	CHR $ (11) Домой Пример 8 PRINT CHR $ (11); POS (Ø); CSRLIN (Ø Ø)	Управляющий код, по которому курсор перемещается в левый верхний угол экрана в точку (Ø, Ø) (аналог HOME)
1Ø	CHR $ (12) Чистка экрана Пример 5 PRINT CHR $ (12); POS (Ø); CSRLIN (Ø Ø)	Управляющий код, по которому производится чистка экрана с установкой курсора а в позицию (Ø, Ø) (аналог SHIFT & GLS)
11	CHR $ (13) Возврат каретки	Управляющий код, по которому курсор перемещается в начало текущей строки (↵)
12	CHR $ (28) CHR $ (29) CHR $ (3Ø) CHR $ (31)	Управляющие коды, по которым курсор перемещается, если это возможно, на 1 позицию «вперед» (→), «назад» (←), «вверх» (↑). или «вниз» (↓)

Ввод и редактирование программ. Некоторые из команд Бейсик-системы (интерпретатора), используемые при вводе и редактировании программ, были рассмотрены ранее. Остальные команды этой группы приведены в таблице 9. Речь идет о командах: автоматической нумерации строк при вводе (AUTO), перенумерации (RENUM) и удаления (DELETE) строк, вывода списка команд на экран (LIST) и принтер (LLIST), чистки оперативной памяти (NEW) и запуска программы на счет (RUN).

Текущим является номер той строки программы, которая последней заносилась (←) в оперативную память. Заметим, что это не обязательно наибольший из номеров строк. В некоторых командах ввода и редактирования допускается ссылка на текущий номер строки. Делается это проставлением на месте соответствующего параметра символа «точка»(.).

```
ПРИМЕРЫ 3Ø
LIST .
LIST 5Ø-.
AUTO .,3
DELETE .
```

Все параметры команд рассматриваемой группы являются целыми неотрицательными числами. При этом номера строк лежат в диапазоне от 0 до 65529.

Таблица 9

Команды интерпретатора, используемые при вводе и редактировании программ

	Команда. Название. Примеры	Выполняемые действия
1	AUTO [n1] [, n2] Автоматическая нумерация строк Примеры AUTO AUTO 100 г AUTO 50,3 г AUTO., 5 г	Automatic — автоматический. По команде AUTO производится включение режима автоматической нумерации строк при вводе от n1 и дальше с шагом n2 (n2⩾Ø). По умолчанию n1=n2=10. Если очередное число k, сгенерированное автоматической нумерацией, совпадает с номером уже находящейся в памяти строки, то на экране индицируется последовательность k«. После ввода (↵) новой строки k происходит следующее. Если за символом * есть хоть один знак, отличный от пробела, то старая строка k в памяти исчезает, а ее место занимает вновь введенная строка. В противном случае прежняя строка сохраняется. Выход из режима AUTO осуществляется по команде: CTRL & STOP
2	RENUM [n1] [, n2] [, n3] Перенумерация строк Примеры	Renumber — перенумеровать. По команде RENUM производится перенумерация программных строк от n2 и дальше с шагом n3, (n3⩾Ø). При этом

Продолжение

	Команда. Название. Примеры	Выполняемые действия
	RENUM RENUM 200, 150, 5 RENUM 100, 200 RENUM 1000	n1 — это новый номер прежней строки n2. По умолчанию n1 = n3 = 10, n2 — номер начальной строки программы. При перенумерации выдаются ошибочные ссылки на номера несуществующих в программе строк
3	DELETE [n1] [— n2] Стирание строк Примеры DELETE 70 — 100 DELETE 50 DELETE. DELETE — 200	Delete — стереть. По команде DELETE из программы удаляются строки от n1 до n2 включительно. По крайней мере одни из параметров n1 и n2 должен присутствовать явно. По умолчанию n1 и n2 равны соответственно наименьшему и наибольшему из номеров строк программы
4	NEW Чистка памяти	New — новый. По команде NEW из оперативной памяти удаляются все программы пользователя и значения переменных
5	LIST [n1] [— n2] Индикация фрагмента программы Примеры LIST LIST 20 — 70 LIST — 50 LIST 40 LIST 500 LIST 100	LIST — список. По команде LIST на экран дисплея выводятся строки от n1 до n2 включительно. Если указанный диапазон строк не умещается на экране целиком, то остается лишь его последняя часть. Пусть α и β — наибольший и наименьший из номеров строк программы. По умолчанию действует такое соглашение. При отсутствии только n1 полагается n1 = α. При отсутствии только n2 полагается n2 = n1. При отсутствии обоих параметров считается n1 = α, n2 = β
6	LLIST [n1] [— n2] Распечатка фрагментов программы	По команде LLIST на печатающее устройство выводятся строки от n1 до n2. По умолчанию действует такое же соглашение, как и в команде LIST
7	RUN [n] Запуск программы Примеры RUN RUN 1000	RUN — запуск, прогон. По команде RUN производится чистка значений переменных и запуск программы на счет со строки n. По умолчанию n равно наименьшему из номеров строк

1.4. Базовые конструкции

Присваивание значений. При запуске программы числовые переменные получают значение 0, а строковые — пусто. В дальнейшем эти значения могут изменяться с помощью команд: присваивания (LET), ввода данных с клавиатуры (INPUT, LINE INPUT), обмена

значениями (SWAP) и считывания из блока данных (DATA, READ, RESTORE). О них и пойдет речь ниже.

LET. Команда LET записывается в виде

$$[\text{LET}]\,\alpha = \beta \qquad (1)$$

где: LET (пусть) — служебное слово; α — простая переменная, элемент массива или MID$-функция; β — выражение.

Слово LET в (1) указывать не обязательно. При выполнении LET вычисляется значение выражения β и результат присваивается α. Если α — числовая переменная, а β — арифметическое выражение, то при необходимости перед присваиванием значение β преобразуется к типу α.

Примеры 31

```
1) 10 X%=7345. 19087#
   20 Y! =574. 123456#
   30 Z! =76767878#
   40 PRINT X%; Y!; Z!
   run
   7345 574. 123 76767900
```

```
2) 10 LET X = Y + Z
   200 A=SIN (X)
   300 Y $ = MID $ (P $, 7, 12)
   400 AB (K) = K+1
```

```
3) 100 MID $ (E $, 5) = Z $ (8).
```

В случае, когда α есть MID$-функция, строковая переменная в ней обязана быть определена заранее. Скажем, в последнем примере значение для E$ должно быть сформировано до выполнения строки 100.

INPUT. Команда INPUT используется для ввода данных с клавиатуры и записывается в виде

$$\text{INPUT}\ [\langle\gamma\rangle;]\ \alpha_1,\alpha_2,\ldots,\alpha_k \qquad (2)$$

где: INPUT (ввод) — служебное слово; γ — последовательность произвольных символов алфавита (кроме запятой); $\alpha_1,\alpha_2,\ldots,\alpha_k$ — список переменных.

При выполнении команды INPUT вычисления приостанавливаются и на экран дисплея выводится поясняющее сообщение γ, если оно имеется, и знак ? : γ?. Далее ЭВМ ждет, пока с пульта клавишного устройства будут введены значения для всех переменных списка $\alpha_1,\alpha_2,\ldots,\alpha_k$. После этого вычисления по программе возобновляются. Вводимые данные $\beta_1,\beta_2,\ldots,\beta_k$ представляют собой числовые или строковые значения. При наборе они отделяются друг от друга запятыми. Тип (числовой, строковый) и количество их должны соответствовать типу и количеству переменных $\alpha_1,\alpha_2,\ldots,\alpha_k$. После на-

жатия на клавишу ↵ значения $\beta_1,\beta_2,...,\beta_k$ последовательно присваиваются этим переменным. Впрочем, вводить данные можно и частями (см. пример 32(1)). Заметим, что если ввести пустую строку или строку из пробелов, то соответствующая переменная сохранит свое прежнее значение (см. пример 32(3)).

При наборе данных до нажатия на клавишу ↵ можно осуществлять их редактирование всеми имеющимися для этого средствами (см. табл. 6).

П Р И М Е Р Ы 32

```
1)  10 INPUT "COPT";X,Y,B$
    run
    COPT? 25          (↵)
    ?? 15,EFG         (↵)
2)  10 INPUT "COPT";X,Y,B$      ?Redo from start
    20 PRINT X;Y;B$             COPT? 25,15,EFG,D
    run                         ?Extra ignored
    COPT? 25                     25  15 EFG
    ?? L
3)  10 A=5:INPUT A:PRINT A
    run
    ? 5
```

Сообщения Бейсик-системы, которые мы имели во втором примере, расшифровываются так:

Redo from start — вводи сначала,
Extra ignored — лишние данные проигнорированы.

Отметим следующие ограничения при вводе данных по команде INPUT. В строковое значение нельзя ввести символ запятой (,) или начинать его с кавычек (″).

LINE INPUT. Команда LINE INPUT используется для ввода с клавиатуры значений для строковой переменной и записывается в виде

LINE INPUT [″γ″;] α (3)

где: LINE, INPUT — служебные слова; γ — последовательность произвольных символов алфавита; α — строковая переменная.

Выполнение команды LINE INPUT проводится аналогично команде INPUT с той лишь разницей, что вводится обязательно строковое и только одно значение, за поясняющим сообщением γ не

индицируется знак вопроса и, наконец, среди вводимых символов могут быть и запятые, и кавычки.

Команда LINE INPUT вместе с функцией VAL может использоваться и для ввода числовых значений.

П Р И М Е Р Ы 33

```
1)  10 LINE INPUT "X=";X$
2)  100 LINE INPUT Y$:Y=VAL(Y$)
3)  10 LINE INPUT "A = ";A$
    20 IF A$="&" THEN 10
    30 IF VAL(A$)=0 THEN 10
```

SWAP. Команда SWAP используется для обмена значениями двух переменных одного типа и записывается в виде

SWAP α, β (4)

где: SWAP (обмен) — служебное слово; α, β — переменные.

При выполнении команды SWAP переменные α и β обмениваются своими значениями. При этом в силу специфики реализации SWAP необходимо явное предварительное определение α и β.

П Р И М Е Р Ы 34

```
1)  10 SWAP X,Y
2)  10 INPUT A$,B$          run
    20 SWAP A$,B$           ? MIX,Y
    30 PRINT A$;B$          YMIX
```

Нетрудно понять, что команда SWAP заменяет три команды присваивания вида:

$$\gamma=\alpha:\alpha=\beta:\beta=\gamma$$

где γ — вспомогательная переменная.

DATA, READ, RESTORE. Удобные средства для хранения данных в теле программы и манипулирования ими предоставляет группа из трех команд: DATA, READ и RESTORE. Остановимся на них подробнее.

Команда формирования блока данных DATA записывается в виде

DATA $\beta_1,\beta_2,...,\beta_k$

где: DATA (данные) — служебное слово; $\beta_1,\beta_2,...,\beta_k$ — список констант.

Строковые константы списка $\beta_1,\beta_2,...,\beta_k$ можно задавать в кавычках и без них. В первом случае в значении константы не должно быть кавычек, а во втором — недопустимы также знаки двоеточия (:) и запятой (,).

ПРИМЕРЫ 35

```
1)  10 DATA 1,2,7,15,"ПАРК"
    20 DATA 6,7,7,100,8
2)  100 DATA АВТОБУС, ТРАМВАЙ
3)  200 DATA "ABC,X:Z"
```

Пусть в программе в строках $n_1,n_2,...,n_s$ ($n_1 \leqslant n_2 \leqslant ... n_s$) имеются команды DATA. При запуске программы на счет по ним создается в памяти так называемый блок данных, который удобно представлять себе в виде линейной последовательности констант так, как это изображено на рисунке 3.

С блоком данных связана некоторая величина ↓, называемая указателем считывания. Она всегда настроена на конкретную константу блока. При запуске программы указатель ↓ «метит» позицию первой константы блока. В дальнейших вычислениях по программе команды DATA участия не принимают.

Команда чтения из блока данных READ записывается в виде

READ $\alpha_1,\alpha_2,...,\alpha_n$

где: READ (читать) — служебное слово; $\alpha_1,\alpha_2,...,\alpha_n$ — список переменных.

При выполнении команды READ из блока данных последовательно извлекаются константы и их значения присваиваются соответственно переменным α_1, α_2, ..., α_n. Выборка начинается с той константы, на которую настроен указатель ↓. При этом сам указатель при каждом считывании смещается по блоку на 1 позицию вправо. Типы значений и переменных (числовые, строковые) должны совпадать. При считывании последней константы блока значение указателя ↓ не определено.

Команда управления указателем считывания из блока данных RESTORE записывается в виде

RESTORE [m] (5)

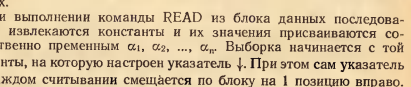


Рис. 3

где: RESTORE (восстановить) — служебное слово; m — номер программной строки с командой DATA.

При выполнении команды (5) указатель ↓ настраивается на ту константу блока данных, которая первой вошла в него из самой «левой» команды DATA строки m. Если параметр m отсутствует, то указатель ↓ настраивается на первую константу блока.

ПРИМЕРЫ 36

```
1)  10 DATA S,ABC,&H6,&O15,&B101
    20 DATA 10E+2,8D-1,"START"
2)  10 DATA 5,6                    run
    20 DATA POLE,9,10               5  6          POLE
    30 DATA "XZ",11                 POLE 9  10 XZ
    40 READ X,Y,Z$
    50 PRINT X;Y;Z$
    60 RESTORE 20
    70 READ L$,A,B,M$
    80 PRINT L$;A;B;M$
```

Передачи управления. По введенной в память ЭВМ программе вычисления проводятся в соответствии с составляющими ее строки командами (слева направо) и в порядке возрастания номеров строк. Однако по подобной линейной схеме решается весьма ограниченное количество задач. Во многих реальных ситуациях желательно уметь нарушать естественный порядок выполнения команд. Такие средства в Бейсике предусмотрены. Некоторые из них и рассматриваются ниже.

GOTO. Команда безусловной передачи управления записывается в виде

GOTO n (6)

где: GOTO (перейти к) — служебное слово; n — номер программной строки.

По команде (6) управление передается строке n. Иными словами, где бы ни находилась в программе команда GOTO n, после нее начинает выполняться первая команда строки n.

Пример. 10 GOTO 1000

IF. Команда IF используется для организации разветвлений вычислительного процесса. Записывается она в виде

$$\text{IF } \sigma \left\{ \begin{array}{l} \text{THEN} \left\{ \begin{array}{l} \text{O1} \\ \text{n1} \end{array} \right\} \\ \text{GOTO n1} \end{array} \right\} \left[\text{ELSE} \left\{ \begin{array}{l} \text{O2} \\ \text{n2} \end{array} \right\} \right]$$

где: IF (если), THEN (то), GOTO (перейти к), ELSE (иначе) — служебные слова; σ — выражение отношения (условие); n1, n2 — номера программных строк; O1, O2 — последовательности команд, разделенные двоеточиями.

Пусть x и y — арифметические или символьные выражения. Условие σ в IF или является простым отношением вида:

$$x<y,\ x>y,\ x<=y,\ x>=y,\ x=y,\ x<\ >y,$$

или строится из них с помощью операций (связок)

OR (или), AND (и), NOT (не) ...

(см. 1.1). Во всех случаях σ может принимать лишь два значения: T (истина) и F (ложь).

При реализации IF выполняется или ее THEN (GOTO)-часть (значение σ равно T), или ее ELSE-часть (значение σ равно F). Если ELSE-части в команде IF нет, то условно можно считать, что она есть, но состоит из «пустой» команды, не производящей никаких действий.

Отметим, что возможны случаи вложения команд IF друг в друга и что IF должна полностью размещаться в одной программной строке.

П Р И М Е Р Ы 37

```
1)  200 IF X<0 THEN 600
    300 IF A=B GOTO 700
    400 IF C<D THEN Y=X^2 ELSE Y=X^3
2)  100 IF A<X AND X<B
        THEN Y=FNT(X):C$="В ОТРЕЗКЕ"
        ELSE Y=FNQ(X):C$="ВНЕ ОТРЕЗКА"
    200 PRINT X;Y,C$
3)  700 IF A=B THEN Y=A
                ELSE IF A<B THEN Y=B-A
                ELSE Y=A-B
```

ON. Общая форма записи команды-переключателя ON следующая.

$$\text{ON } \beta \left\{ \begin{matrix} \text{GOTO} \\ \text{GOSUB} \end{matrix} \right\} n_1,\ n_2,\ \ldots,\ n_k$$

где: ON (на), GOTO, GOSUB (идти к подпрограмме) — служебные слова; β — арифметическое выражение; n_1, n_2, ..., n_k — номера программных строк.

Выполнение команды ON начинается с вычисления целой части p значения выражения β. Далее, если p∈{1, 2, ..., k}, то управление переходит к строке n_p. При p=0 или p>k управление передается команде, следующей за ON. Другие значения p приводят к сообщению об ошибке.

Параметры GOSUB и GOTO в ON используются в зависимости от того, являются или нет величины n_1, n_2, ..., n_k номерами строк подпрограмм. В первом из этих случаев после выполнения соответствующей подпрограммы управление передается команде, расположенной за ON.

П Р И М Е Р Ы 38

```
10  INPUT X                                  run
20  ON X GOTO 200,300,400:GOTO 10            ? 2
200 PRINT X+200:GOTO 10                       302
300 PRINT X+300:GOTO 10                       ? 3
400 PRINT X+400:GOTO 10                       403
```

Циклы. С помощью команд условной передачи управления IF нетрудно организовать повторные выполнения определенной совокупности команд, называемой в этом случае циклом. Например, для вычисления суммы $\sum_{k=1}^{100} k/(k^2+3)$ можно использовать следующий фрагмент:

```
10 K=1:S=0                                   run
20 S=S+K/(K^2+3)                              4.0320702699077
30 K=K+1
40 IF K<=100 THEN 20
50 PRINT S
```

При запуске этой программы команды 20—40 будут исполнены по 100 раз. Они и составляют цикл.

Понятие *цикла* является одним из основополагающих понятий информатики. Программы, содержащие циклы, называются *циклическими*. Цикл, не содержащий внутри себя других циклов, называется *простым*. В противном случае его называют *кратным (сложным, составным)* циклом.

Для эффективной организации циклических программ в Бейсике предусмотрены специальные средства. Речь идет о команде FOR, общая форма записи которой приведена на рисунке 4.

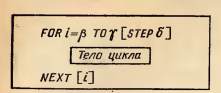


Рис. 4

Здесь: FOR (для), TO (до), STEP (шаг), NEXT (следующий) — служебные слова; i — числовая переменная; β, γ, σ — арифметические выражения.

Общепринята следующая терминология:

i — параметр цикла,
β — начальное значение параметра цикла,
γ — конечное значение параметра цикла,
σ — шаг цикла (σ≠0, а по умолчанию σ=1),
FORi=βTOγ [STEPσ] — заголовок цикла,
NEXTi — окончание цикла,

команды между заголовком и окончанием образуют тело цикла. Выполнение команды FOR проводится так.

1. Вычисляются и запоминаются значения выражений β, γ и σ. Пусть они равны соответственно b, e и h.
2. Параметр цикла i получает начальное значение b: $i=b$.
3. Выполняются команды тела цикла.
4. Параметр i получает приращение h: $i=i+h$.
5. Проверяется условие $(e-i)h \geqslant 0$. Если оно истинно, то следующим выполняется шаг 3, иначе выполняется шаг 6.
6. Команда цикла прекращает свою работу.

Заметим, что хотя бы один раз тело цикла будет выполнено при любых β, γ и σ. Значения выражений β, γ и σ внутри цикла изменить нельзя, значение параметра i — можно. Вход в цикл реализуется только через его заголовок. Возможен досрочный выход из цикла не через его окончание, а по командам IF, GOTO... . Вложенные циклы могут обслуживаться одной строкой окончания. Например, NEXT K, J, I. Одиночный параметр цикла в NEXT указывать не обязательно.

```
П Р И М Е Р  39                    2)  300 FOR I=1 TO 20
1)  100 S=0                            310 FOR J=1 TO 30
    110 FOR K=1 TO 100                 320 INPUT MAS(I,J)
    120 S=S+A(K)                       330 NEXT J,I
    130 NEXT K
3)  100 INPUT A,B,H
    110 FOR X=A TO B STEP H
    120 PRINT X;EXP(SIN(X))
    130 NEXT
```

Вывод данных. В этом пункте речь пойдет о командах вывода данных на символьные экраны 0 и 1 (PRINT, PRINTUSING),

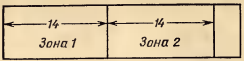


Рис. 5

графические экраны 2 и 3 (PRINT#, PRINT#, USING) и принтер (LPRINT, LPRINTUSING). Впрочем, более подробно последний случай описан в пункте 3.

Вывод данных на символьные экраны 0 и 1.

PRINT. Условно разобьем поле ввода фона экрана дисплея на зоны по 14 позиций каждая. Зоны символьных экранов изображены на рисунке 5. Поскольку максимальная ширина фона не превосходит 40 позиций, то количество полных зон не может быть более 2.

По команде PRINT (печатать) данные индицируются на экранах 0 или 1. Возможности управления размещением выводимой информации ограничены. Общая форма записи PRINT следующая:

$$\text{PRINT } [\beta_1 r_1 \beta_2 r_2 ... \beta_k [r_k]] \qquad (7)$$

где: PRINT — служебное слово; $\beta_s (s=1, 2, ..., k)$ — выражения, значения которых должны быть проиндицированы; $r_s (s=1, 2, ..., k)$ — указатели формата, разделяющие выражения. Ими могут быть запятая, точка с запятой или последовательности этих символов. Для r_k допускается и пробел.

При выполнении команды PRINT все значения $\beta_s (s=1, 2, ..., k)$ выводятся, начиная с текущей позиции курсора. Причем индикация каждого символа смещает курсор на 1 позицию «вперед». Разделители r_s действуют так: каждая запятая переводит курсор в начало следующей полной зоны, а точки с запятой на него влияния не оказывают. Разделитель r_k, равный пробелу, смещает курсор в начало первой зоны следующей строки. Так же действует команда PRINT без параметров. За каждым выведенным числовым значением всегда помещается один пробел.

При вводе слово PRINT можно заменять символом ?.

```
П Р И М Е Р Ы  40
1)  10 PRINT "ПРО";"БА";7;"SMIT";4;55
    run
    ПРОБА 7 SMIT 4  55
2)  100 PRINT "ЗАРПЛАТА";200;"РУБ"
    run
    ЗАРПЛАТА 200 РУБ
              ↖ пробел для знака
```

```
3)  30 X=7                          run
    40 PRINT X,X^3                  7           343
    50 PRINT X;X^3                  7  343
    60 PRINT X;-X^3                 7 -343
```

PRINTUSING. С помощью PRINTUSING (печать по шаблону) можно редактировать выводимую на экран информацию, располагая ее требуемым образом. Общая форма записи этой команды следующая:

$$\text{PRINTUSING } t;\ \beta_1[r_1\beta_2 r_2 \ldots \beta_k r_k] \qquad (8)$$

где: PRINTUSING — служебное слово; $\beta_s (s = 1, 2, \ldots, k)$ — выражения, значения которых выводятся; t — строковое выражение, задающее формат (шаблон) вывода; $r_s (s = 1, 2, \ldots, k)$ — разделители выражений. Ими могут быть запятая или точка с запятой. Для r_k допускается и пробел.

Некоторые символы t являются управляющими и специальным образом влияют на формат вывода. Другие символы t «нейтральны» и просто переносятся в выводимые значения. Тип разделителей r_s на вывод не влияет.

Ниже перечислены основные управляющие символы шаблонов и даны поясняющие примеры.

1) ! — вставка первого символа строкового значения. Количество знаков ! в шаблоне должно совпадать с количеством элементов вывода в команде.

```
10 X$="ZZ":Z$="+"                   run
20 PRINTUSING "A!B-C!D/E";X$,Z$     AB-C+D/E
```

2) \␣ ␣...␣\ — вставка первых n+2 символов строкового значения (при n пробелах). Количество пар \...\ в шаблоне должно совпадать с количеством элементов вывода в команде.

```
10 Y$="1234567890":Z$="КОМПЬЮТЕР"
20 PRINTUSING "A\ \B\   \C\     \D";Y$,Z$;Y$
run
A123BKOMПC12345D
```

3) & — вставка полного строкового значения. Количество символов & в шаблоне должно совпадать с количеством элементов вывода в команде.

```
10 Y$="БЕЙСИК":FO$="MSX-&"          run
20 PRINTUSING FO$;Y$                MSX-БЕЙСИК
```


4) # # # — примеры форматов для вывода чисел в фиксированной форме.
   # #.#

```
1)  10 ?USING "## ";25,-1,2;34;573
    run
    25 -1  2 34 %573
2)  100 DATA 7,4.576,45.8
    110 FOR X=1 TO 3:READ Z
    120 PRINTUSING "#.##РУБ";Z
    130 NEXT X
    run
    7.00РУБ
    4.58РУБ
    %45.80РУБ
```

5) + # # #.# — примеры форматов для вывода чисел в фикси-
   # #.# # # + рованной форме с указанием знака + (плюс)
   или — (минус) в начале или в конце записи.

```
1)  10 F$="+#.##   "
    20 PRINTUSING F$;-1.852,3.46
    run
    -1.85  +3.46
2)  10 F$="#.##+   "
    20 PRINTUSING F$;-1.857,3.46
    run
    1.86-  3.46+
```

6) #.# # — — пример формата для вывода чисел с указанием знака минус в конце отрицательных значений.

```
100 F$="##.#-   "
200 ?USING F$;-75.3,14,-77.7,756
run
75.3-  14.0   77.7-  %756.0
```

7) **# # # # — пример формата для вывода чисел с забиванием пустого пространства перед ними символами *.

```
10 PRINTUSING "%%###.#  ";7,945,1841.1
run
%%%7.0  %%945.0  %1841.1
```

8) #.##^^^^ — пример формата для вывода чисел в плавающей форме.

```
10 ?USING "#.###^^^^  ";5844,2E-03,5
run
0.584E+04  0.200E-02  0.500E+01
```

9) ##,.#### — пример формата для вывода чисел в фиксированной форме с разбиением записи их целой части на грани по 3 цифры в каждой.

```
10 GR$="##########,.####"
20 ?USING GR$;87654321.77777
run
 87,654,321.7778
```

Обратите внимание на то, что при необходимости числа перед выводом округляются и что если число шире формата, то оно все равно выводится, но перед ним ставится знак предостережения %. Далее разделители запятая (,) и точка с запятой (;) в списке вывода никоим образом не меняют положения курсора. Несколько значений, выводимых по одному шаблону, располагаются друг за другом без дополнительных пробелов.

Форматы могут иметь весьма замысловатую структуру и служить для вывода сразу нескольких значений.

ПРИМЕРЫ 41

1)
```
10 T$="НАРЯД-###  СУММА -~~ ###.## Р"
20 ?USING T$;23,172.66
run
НАРЯД- 23  СУММА --- 172.66 Р
```

2)
```
100 W$="##.# + ##.#%i"
200 ?USING W$;7.7,56.345
run
 7.7 + 56.3%i
```

Вывод данных на графические экраны 2 и 3. Для вывода символьной информации на графические экраны требуется предварительное «открытие» устройства вывода "GRP" (см. п. 4.4.). Сделать это можно, например, так

OPEN "GRP:" AS #1

PRINT # 1. Общая форма записи команды PRINT #1 следующая:

PRINT #1, [$\beta_1 r_1 \beta_2 r_2 ... \beta_k [r_k]$] (9)

Здесь все параметры имеют тот же смысл, что и в команде PRINT. Выполняются команды (7) и (9) также одинаково. Исключением является лишь тот факт, что в данном случае при индикации текста прежняя информация с графического экрана не стирается и без специальных предосторожностей возможны нежелательные наложения символов и рисунков.

ПРИМЕР 42

```
10 COLOR 1,15,7:SCREEN 2
20 OPEN "GRP:"AS#1
30 PRINT #1,"графический экран"
40 FOR K=1 TO 3000:NEXT
```

PRINT # 1. USING. Записывается данная команда в виде:

PRINT #1,USING t; $\beta_1 [r_1 \beta_2 r_2 ... \beta_k r_k]$ (10)

Здесь все параметры имеют тот же смысл, что и в команде PRINTUSING. Выполнение (8) и (10) также одинаково, за исключением отсутствия в рассматриваемом случае чистки очередного знакоместа перед выводом очередного символа (см. PRINT #1).

Вывод данных на принтер. Рассмотрим два возможных варианта вывода данных на бумажную ленту печатающего устройства, которую удобно считать условно разбитой по ширине на зоны по 14 позиций в каждой.

Способ 1. Этот прием вполне идентичен выводу символов на графические экраны. Он возможен при предварительном «открытии» устройства печати по команде

OPEN "LPT:"AS#1

Дальнейшее использование PRINT#1 и PRINT#1,USING производится обычным образом.

Предлагаемый способ хорош тем, что фактически позволяет писать программы, заранее не ориентированные на конкретное устройство вывода.

Способ 2. По командам

LPRINT $[\beta_1 r_1 \beta_2 ... \beta_k [r_k]]$

LPRINTUSING t; $\beta_1[r_1\beta_2 ... \beta_k r_k]$

организуется вывод информации на принтер с управлением, аналогичным тому, который применялся в соответствующих командах (7) и (8).

ПРИМЕР 43. Печать таблицы шестнадцатиричных кодов ASCII символов БЕЙСИКА

```
 10 LPRINT "    ";
 20 FOR S=2 TO 15
 30 LPRINT HEX$(S);"  ";
 40 NEXT S
 50 LPRINT:LPRINT
 60 FOR K=0 TO 15
 70 LPRINT HEX$(K);"   ";
 80 FOR S=2 TO 15
 90 R=16*S+K
100 LPRINT CHR$(R);"  ";
110 NEXT S
120 LPRINT
130 NEXT K
```


Результатом выполнения приведенной программы является таблица 1.

Подпрограммы. Прежде всего заметим, что подпрограммы — это специальным образом оформленные группы строк. Подпрограммы бывают двух типов: *стандартные* и *нестандартные*. Первые из них входят в состав математического обеспечения систем. Нам часто приходится неявным образом обращаться к ним, когда, например, вводим данные в память, печатаем результаты счета, вычисляем значения стандартных функций и т. п.

Нестандартные подпрограммы составляются самим пользователем и являются фактически фрагментами его программ. В этом пункте рассматриваются подобные подпрограммы, организуемые с помощью команд GOSUB — RETURN. Общая форма записи команды обращения к подпрограмме GOSUB n, где: GOSUB — служебное слово; n — номер программной строки.

При выполнении GOSUB (GO SUBROUTINE — идти к подпрограмме) происходит передача управления в подпрограмму строке с номером n. Подпрограмма обязана завершаться командой RETURN (возврат), с помощью которой организуется возврат к команде, следующей за GOSUB. Войти в подпрограмму можно в любом месте, но не внутрь цикла. Иными словами, фиксированного начала она не имеет.

Из рисунка 6 можно уяснить механизм передач управления подпрограмме при наличии в основной программе нескольких команд GOSUB.

Возможны последовательные обращения из одних подпрограмм в другие с достаточной глубиной подобного вложения. Рисунок 7 иллюстрирует схему вызова подпрограмм A_k (k = 1, 2, ..., 9) с максимальной глубиной вложения 4.

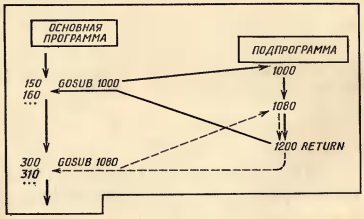


Рис. 6

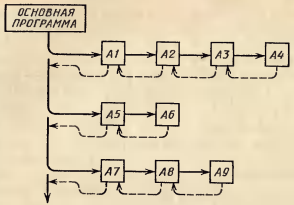


Рис. 7

Пример. Составить программу, по которой методом половинного деления (вилки) на отрезке [A, B] с точностью E определяется какой-либо вещественный корень уравнения

$$F(x) = 0, \qquad (11)$$

где F (x) непрерывна на [A, B] и F (A)·F (B) <0.

Пусть, например,

$$F(x) = \cos(e^x) - \sin(\pi^x)$$

A=1, B=3, E=0,00001.

В приведенной ниже программе ядро составляют строки 90—110. При счете из основной программы производится многократное обращение к подпрограмме (строки 200—230), в которой при текущем значении x вычисляется величина y=F (x). Каждому уравнению (11) соответствует своя подпрограмма вида:

```
220 Y=F (X)
230 RETURN
```

Заметим, что вычисления y=F (x) не обязательно должны быть организованы в одной программной строке.

```
П Р И М Е Р  44
10 '  МЕТОД  'ВИЛКИ' ДЛЯ УРАВНЕНИЯ  F(X)=0
20 ' ==========================================
30 CLS:PRINT "[A,B] - отрезок, E - точность
```



```
33 U$="корень- ###.####, точность- ##^^^^"
40 INPUT "введите тройку чисел A,B,E";A,B,E
50 X=B:GOSUB 200:F2=SGN(Y):IF Y=0 THEN 120
60 X=A:GOSUB 200:F1=SGN(Y):IF Y=0 THEN 120
70 IF F1*F2>0 THEN 30
80 '-----------------------------------ядро
90 X=(A+B)/2:GOSUB 200:F2=SGN(Y)
95 IF Y=0 THEN 120
100 IF F1*F2<0 THEN B=X ELSE A=X:F1=F2
110 IF B-A>E THEN 90  '-----------------ядро
120 PRINTUSING U$;X,E
130 END
200 '   вычисления значений функции F(X)
210 '-----------------------------------------
220 Y=COS(EXP(X))-SIN(3.14156^X)
230 RETURN
```

Замечание. Строки 10, 20, 80, 200 и 210 в примере 44 являются комментариями и не влияют на ее выполнение.

Массивы. Ранее мы имели дело с переменными, которые характеризовались именем, типом и в любой момент времени единственным текущим значением. Подобные переменные называют простыми. Наряду с простыми переменными в Бейсике допустимы и массивы. Фактически массив представляет собой индексированную одним или несколькими индексами последовательность простых переменных одного типа, обозначаемых одним именем. В программах массивы объявляются, или описываются командой DIM $\gamma_1, \gamma_2, \ldots, \underline{\gamma_k}$, где: DIM (*dimention* — размеры) — служебное слово; γ_s (s=1, k) — компоненты объявления.

Каждый компонент объявления определяет имя (или имя и тип) массива, а также верхние границы его индексов. Нижние границы по умолчанию равны 0. Верхние границы индексов задаются целыми неотрицательными константами или числовыми переменными. Их значения ограничиваются только размерами оперативной памяти. Если массив не объявлен в DIM, верхние границы всех его индексов принимаются равными 10.

Командой DIM можно объявлять и простые переменные.

Пример 45.

1) 10 DIM C% (100)

Имя массива	— C% ( )
Тип	— целочисленный
Размерность	— одномерный
Имена элементов	— C% (0), C% (1), ... C% (100) C% (X), C% (A+B) и т. п.
Количество элементов	— 100+1=101

2) 10 DIM A (5,20)

Имя массива	— A
Тип	— числовой, двойной точности
Размерность	— двумерный
Имена элементов	— A (0, 0), A (0, 1) ...
Количество элементов	A (5, 20), A (I%, I%) и т. п. — (5+1) (20+1)=126

3) 10 DIM C! (7, 8, 9)

Имя массива	— C!
Тип	— числовой, одинарной точности
Размерность	— трехмерный
Имена элементов	— C! (3, 2, 1), C! (X, Y, Z) и т. п.
Количество элементов	— (7+1) (8+1) (9+1)=720

4) 10 DIM E$ (20)

Имя массива	— E$
Тип	— строковый
Размерность	— одномерный
Имена элементов	— E$ (0), E$ (1), ... E$ (20)
Количество элементов	— 20+1=21

При выполнении команды DIM фиксируется наличие соответствующих массивов и проводится начальная инициализация их числовых элементов нулями, а строковых — нуль-строкой (“”).

ПРИМЕРЫ 46

```
1)  10 DIM A!(20),B%(20,30),C(3,3,3),E%
2)  50 DIM A(30)
    60 FOR K=1 TO 30
    70 INPUT A(K)
    80 NEXT K
```

Тип массива можно определять не только в самом имени знаками $, %, # и !, но и в команде DEF фиксаторами STR, INT, DBL и SGN (см. пункт 1.1).


```
ПРИМЕР 47
10 DEFSTR L:DIM L(100)
20 L(33)="строковое значение"
30 PRINT L(33)
run
строковое значение
```

Высвободить занятую под массив память можно по команде

ERASE $\gamma_1, \gamma_2, ..., \gamma_k$

где: ERASE (стирать) — служебное слово; γ_s ($s=\overline{1,k}$) — имена массивов (без скобок).

Высвобождаемые имена могут быть объявлены новой командой DIM. Команда ERASE является единственным средством управления оперативной памятью при ее нехватке.

ПРИМЕРЫ 48

```
1)  10 ERASE A!,B%,C#,D
```

```
2)  10 DIM A(50)
    20 FOR K=1 TO 50
    ...  ...  ...
    200 NEXT K
    210 ERASE A
```

1.5. Дополнительные возможности языка

Управление ресурсами памяти. Персональный MSX-компьютер имеет небольшой объем оперативной памяти — 64 Кбайта. Поэтому полезной оказывается информация о распределении ресурсов памяти под различные цели и сведения о наличии и объеме в ней свободных мест в любой момент времени. На рисунке 8 представлена карта распределения оперативной памяти с указанием шестнадцатиричных адресов. Для недискового Бейсика α=F3B0, а для дискового значение α зависит от количества подключаемых дисководов. Из этой карты ясно, что вся память разбивается на три основные части: ПЗУ (постоянное запоминающее устройство), ОЗУ (оперативное запоминающее устройство) и рабочая системная область. В первой из них располагается Бейсик-интерпретатор, использующий при работе системную область. Во второй — программы пользователя, значения простых переменных и элементов массивов.

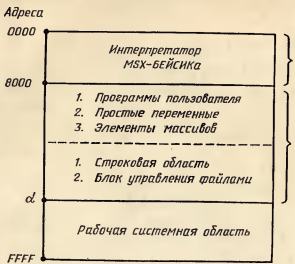


Рис. 8

Иногда требуется иметь предварительную оценку объема памяти, необходимой для размещения значений всех переменных конкретной программы. Сделать ее поможет таблица 10, в которой указы-

Таблица 10

Простые переменные	Байты	Элементы массивов	Байты
Целая	3+2	Целая	2
Одинарной точности	3+4	Одинарной точности	4
Двойной точности	3+8	Двойной точности	8
Строковая	6+L	Строковая	3+L

вается колнчество байт памяти, отводимых под переменные разного типа. Через L там обозначена длина значения строковой переменной.

В Бейсике имеются средства для получения сведений о наличии свободного места в ОЗУ (функция FRE) и управления размером строковой области (CLEAR). Остановимся на них подробнее.

FRE. Записывается функция FRE (*free* — свободный) в одной из следующих двух форм: FRE (0), FRE (""). В первом случае FRE имеет значением количество байт свободной части памяти в ОЗУ без учета строковой области. Во втором — сколько свободного места в байтах имеется в строковой области. Кроме того, FRE ("") проводит так называемую «сборку мусора», реорганизуя расположенную в ОЗУ информацию и, по возможности, увеличнвая свободную память. Однако следует иметь в виду, что «сборка мусора» выполняется медленно.

ПРИМЕРЫ 49

```
1) PRINT FRE(0)
run
 28815
2) 100 IF FRE(0)>20000 THEN 250
```

Следующий пример дает более полную иллюстрацию использования функции FRE, и результат его прогона подтверждает данные таблицы 10. Обратим внимание, что значения строковых констант "десять букв", "К" и "фокус" берутся из самой программы и, таким образом, не занимают дополнительного строкового пространства.

ПРИМЕР 50

```
10 GOSUB 120
20 X=451:GOSUB 120
30 Z#=7.5:GOSUB 120
40 Y!=555:GOSUB 120
50 W%=111:GOSUB 120
60 U$="десятьбукв":GOSUB 120
70 D$="К":GOSUB 120
80 DIM E$(150):GOSUB 120
100 FOR K=1 TO 150:E$(K)=CHR$(K):NEXT
105 GOSUB 120
110 E$(1)="фокус":GOSUB 120:END
120 PRINT FRE(0),FRE(""):RETURN
run
```

28586	200	28546	200
28575	200	28540	200
28564	200	28079	200
28557	200	28068	50
28552	200	28068	51

Функция FRE (0) с успехом может применяться и для защиты программы от чьих-либо попыток ее модифицировать. В некоторую точку программы вставляется дополнительная команда

IF FRE(0) $<>$ α THEN NEW

проверяющая, соответствует ли реальное свободное пространство тому, которое должно быть на самом деле. При FRE (0)≠α IF производит чистку оперативной памяти, то есть организует самоуничтожение программы и значений переменных. При этом α для своей программы необходимо вычислить с учетом добавляемой команды IF.

CLEAR. Строковая область ОЗУ используется для размещения строковых значений. По умолчанию под нее отводится 200 байт. Однако размеры этой области можно менять командой CLEAR (чистить). В общем виде она записывается так: CLEAR [n], где: CLEAR — служебное слово; n — целое число.

CLEAR без параметров производит «чистку» в ОЗУ всех числовых и строковых переменных, элементов массивов и определенных пользователем функций DEFFN.

```
П Р И М Е Р  51

10 M=9

20 T$=STRING$(5,"+")

30 PRINT M;T$,FRE(0);FRE("")

40 CLEAR

50 PRINT M;T$,FRE(0);FRE("")

run
 9 +++++          28720  195
 0                28737  200
```

CLEAR с параметром n, кроме «чистки», объявляет размер строковой области (см. рис. 8) для расположения строковых значений.

```
П Р И М Е Р  52. ( без CLEAR n )

10 T$=SPACE$(198)

20 B$=STRING$(2,"#")

30 PRINT B$

40 X$=STRING$(1,"L")

50 PRINT X$

run

##

Out of string space in 40
```


При прогоне примера получили сообщение об отсутствии места в строковой области. Действительно, 200 байт, отведенных для нее по умолчанию, оказались уже исчерпанными.

```
П Р И М Е Р  53. ( с CLEAR n )

10 CLEAR 300                    run

20 A$=SPACE$(255)               LLLLL          40

30 B$=STRING$(5,"L")

40 PRINT B$,FRE("")
```

Дополнительные команды и функции. Ниже речь пойдет о некоторых важных дополнительных морфологических компонентах Бейсика, синтаксис и семантика которых по тем или иным причинам не были описаны ранее. Остановимся на командах комментария (REM), завершения вычислений (END), изменения значений в ячейках оперативной памяти (POKE) и видеопамяти (VPOKE) и функциях: ввода одиночных символов (INKEY$), ввода последовательности символов (INPUT$), определения и изменения значений таймера (TIME), получения содержимого ячейки оперативной памяти (PEEK) и видеопамяти (VPEEK), программного распознавания типа клавиши управления курсором (STICK).

REM. Различные сообщения t, которые облегчают понимание текста программы, можно задавать командой REM t, где: REM (*remark* — замечание) — служебное слово; t — произвольная последовательность символов алфавита.

На ход вычислений команды REM никакого влияния не оказывают.

Другой способ включения в текст программы комментариев — это использование апострофа. Все, что стоит в команде за апострофом, рассматривается как поясняющее сообщение.

```
П Р И М Е Р Ы  54

10 REM БИНАРНАЯ СОРТИРОВКА

100 REM "КОНТРОЛЬНАЯ РАБОТА #3"

1000 D=PEEK(0+2 ' это седьмое ответвление
```

Следует помнить, что не все комментарии одинаково полезны. Многие из них, в особенности написанные начинающими программистами, представляют собой лишь «шум» и не несут никакой полезной информации. Во всех случаях комментарии должны формироваться вместе с самой программой, разъяснять ее отдельные сложные фрагменты, подчеркивать использование нетривиальных идей и отражать этапы реализуемого вычислительного процесса. При наличии хороших комментариев значительно упрощаются

процессы тестирования, отладки, сопровождения и модификации программ.

END. По команде END (конец) вычисления по программе прекращаются, закрываются все файлы и компьютер переходит в режим непосредственного счета.

ПРИМЕРЫ 55

```
1)  10 INPUT A
    20 PRINT SIN(A),COS(A),TAN(A)
    30 END
2)  IF C=0 THEN END ELSE Y=X-2
```

Команда END не является обязательной для программы. Вычнслення прекращаются также после выполнения последней команды строки с наибольшим номером, если, разумеется, эта команда не предусматривает передачи управления.

INKEY$. Для ввода данных с клавиатуры ранее мы имели две команды INPUT и LINE INPUT. Для ввода отдельных символов можно также использовать и функцию INKEY$.

В момент нажатия на какую-либо клавишу INKEY$ получает строковое значение, равное вводимому по ней символу. Если клавиши не нажимаются, то значением INKEY$ является нуль-строка (""). Специфика ввода по INKEY$ следующая. На экран не выводится ни знак вопроса, ни эхо символа, и, что самое существенное, не требуется нажатия на клавишу «ввод строки» (↵).

ПРИМЕР 56

```
10 W$=INKEY$                    run
20 IF W$="" THEN 10             A
30 PRINT W$:GOTO 10             E
                                ...   ...
```

По функции INKEY$ можно получать десятичные коды не только «печатаемых символов», но и коды управляющих и значения функциональных клавиш (см. таблицу 11).

ПРИМЕР 57

```
10 W$=INKEY$
20 IF W$="" THEN 10
30 PRINT ASC(W$):GOTO 10
```


Таблица 11

	Клавиши	Коды
1	↵	13
2	ESC	27
3	TAB	9
4	BS	8
5	SELECT	24
6	CLS HOME	11
7	CLS HOME & SHIFT	12
8	INS	18
9	DEL	127
10	←	29
11	→	28
12	↑	30
13	↓	31
14	F1—F10	Коды символов значения функционального ключа

INPUT$. Функция INPUT$ (n), где n — арифметическое выражение, целая часть значения которого лежит в диапазоне от 1 до 255, прерывает вычисления до получения с клавиатуры n последовательных символов. Они и становятся значением функции.

ПРИМЕРЫ 58

```
1)  10 W$=INPUT$(3)
    20 PRINT W$:GOTO 10
2)  100 X$=INKEY$:IF X$="" THEN 100
    200 X$=INPUT$(1)
```

В последнем примере строки 100 и 200 идентичны, не считая того, что по INPUT$ на экран выводится курсор.

Функцию INPUT$ можно также использовать и для получения последовательных десятичных кодов ASCII символов и управляющих клавиш и значений функциональных клавиш (см. табл. 11).

ПРИМЕР 59

```
10 W$=INPUT$(2)
20 FOR K=1 TO 2
30 PRINT ASC(MID$(W$,K,1))
40 NEXT:GOTO 10
```

TIME. MSX-компьютер обладает счетчиком, который называется *таймером* и запускается автоматически при включении машины. Таймер принимает следующие последовательные значения: 0, 1, 2, ..., 65535. Дойдя до 65535, он снова восстанавливается со значения 0 и т. д. Каждые 50 «тиков» таймера соответствуют одной секунде времени. Значения таймера и функции TIME (время) всегда совпадают. Впрочем, имеется возможность управлять этими значениями по команде

TIME=α (1)

где α — арифметическое выражение. При выполнении (1) TIME получает значение, равное INT (α).

П Р И М Е Р Ы 60

```
1) 10 PRINT TIME
   200 X=TIME
   500 Y=RND(-TIME)
2) 60 TIME=0
3) 10 INPUT T:TIME=T
   20 FOR M=1 TO 1000:NEXT
   30 PRINT TIME;:GOTO 20
   run
   ? 0
    138  276  415  554  693 ...
   run
   ? 64982
    65120  65259  65397  0  139  277 ...
4) 100 TIME=0
   ...   ...
   900 PRINT "Время счета = ";TIME/50
```

Из-за небольшой разрядности таймера его обновление происходит довольно быстро — приблизительно каждые 21,8 минуты. Учет более длительных интервалов времени необходимо производить программным путем.

POKE. По команде POKE t, α, где POKE (совать, пихать) — служебное слово; t — арифметическое выражение, целая часть значения которого (T) задает адрес ячейки памяти (конкретного байта);

α — арифметическое выражение, целая часть значения которого (A) лежит в пределах от 0 до 255.

При выполнении POKE значение A записывается в ячейку с адресом T.

П Р И М Е Р 61

```
10 SCREEN 0:WIDTH 39
20 PRINT "КРАЙ ПОЛЯ"
30 POKE &HF3B0,20
40 FOR S=1 TO 3000:PRINT "*";:NEXT
50 WIDTH 39
```

Команда строки 30 устанавливает ширину экрана в 20 позиций. При этом «чистки» изображения не происходит. Последующие изменения изображения на фоне организуются только на установленном поле ввода и не касаются его краев.

П Р И М Е Р 62

```
1) 200 POKE &HFCAB,255:RETURN:' заглавные
2) 300 POKE &HFCAB,0  :RETURN:' малые
3) 400 POKE &HFCAC,255:RETURN:' русские
4) 500 POKE &HFCAC,0  :RETURN:' латинские
```

Здесь демонстрируются возможности программного управления режимами: CAPS — ввод заглавных и строчных букв и РУС — ввод русских и латинских букв. Если, например, в вашей программе зафиксирован фрагмент примера 62(4) и произведено обращение

50 GOSUB 300: GOSUB 500

то в дальнейшем с клавиатуры будут вводиться малые латинские буквы до тех пор, пока программно или с пульта не будут установлены новые режимы. Заметим, что управление в примере 62(4) не изменяет текущего состояния индикаторных лампочек на соответствующих кнопках.

В примере 63 команда POKE применяется для формирования новых программных строк в тексте программы в процессе ее выполнения. Это дает возможность по запросу с экрана дисплея вводить с клавиатуры не только числовые параметры задач, но и аналитические выражения. Можно, например, составить программу приближенного вычисления интегралов Римана по формуле Симпсона и запрашивать с экрана такие параметры: 1) нижний предел интегрирования; 2) верхний предел интегрирования; 3) подынтегральную функцию; 4) точность вычислений.

```
ПРИМЕР 63
10 ' ввод программных строк:'----------
20 LINE INPUT A$
30 N$="999 ":L=39-LEN(N$)-1-8-1
40 IF LEN(A$)>L THEN A$=LEFT$(A$,L)
50 A$=N$+A$+CHR$(13)+"goto 150"+CHR$(13)
60 FOR K=1 TO LEN(A$)
70 POKE &HFBF0+K-1,ASC(MID$(A$,K,1))
80 NEXT K
90 POKE  &HF3FA,&HF0
100 POKE &HF3FB,&HFB
110 D=&HF0+LEN(A$)
120 POKE &HF3F8,D MOD 256
130 POKE &HF3F9,&HFB+D\256:'----------
140 END
150 '--------передача управления---------
        ... ...
```

Несколько слов о том, как выполняется программа. Сначала по запросу LINE INPUT из строки 20 вводится произвольная последовательность A$ команд Бейсика. Наиболее часто A$ есть одиночное присваивание типа Y=F(X). Далее, строки 30—80 формируют в буфере клавиатуры (адреса с FBF0 по FC17) строку следующей структуры:

999 A$ CHR$ (13) GOTO 150 CHR$ (13).

Для чего это делается? Дело в том, что при переходе компьютера по END в режим непосредственного счета сразу же начинают выполняться предписания из буфера клавиатуры. В нашем случае до первого кода CHR$ (13) они равнозначны вводу отредактированной строки 999 в оперативную память и далее, до второго CHR$ (13) — передаче управления на строку 150.

В строках 90—130 устанавливаются так называемые «флажки», то есть по определенным адресам размещается необходимая служебная информация об адресах начала и конца записи в буфере клавиатуры.

Остановимся еще на двух моментах. Во-первых, длина буфера клавиатуры составляет всего лишь 39 байт, что существенно ограничивает и длину создаваемых с ее помощью программных строк. Во-вторых, ко всяким самомодификациям программ следует относиться весьма осторожно. Более того, многими специалистами она вообще отвергается как прием «хорошего» программирования.

PEEK. Функция PEEK (заглянуть) записывается в виде PEEK(t), где t — арифметическое выражение ($0 \leqslant INT(t) \leqslant 65535$). Значением PEEK является содержимое ячейки INT(t). Наиболее часто PEEK используется для экспериментов с данными, хранящимися в ОЗУ.

VPOKE. В команде VPOKE t,α параметры t и α имеют тот же смысл, что и в POKE, и действия этих команд во многом схожи. Только здесь t указывает адрес T ($0 \leqslant T \leqslant 16383$) в видеопамяти (ВП), куда и записывается байт данных $A = INT(\alpha)$.

Остановимся на одном из возможных использований рассматриваемой команды. В видеопамяти при выполнении SCREEN0 или SCREEN1 инициируются шаблоны всех стандартных символов Бейсика, по которым впоследствии и организуется их индикация на экране. Поэтому если после назначения экрана провести изменение шаблона конкретного символа в ВП, то другим станет и сам выводимый символ. Так появляется возможность создавать отдельные новые знаки, национальные алфавиты, стилизованные буквы и т. п. Поясним, как это делается.

Под шаблон каждого стандартного символа с кодом ASCII $\geqslant 32$ в ВП отводится 8 байт, начиная с адреса

$$T = ASC(\text{"символ"}) * 8 + \Delta$$

где

$$\Delta = \begin{cases} 0 & \text{— для экрана 1} \\ 2048 & \text{— для экрана 0.} \end{cases}$$

Поэтому для изменения, например символа "@" на "ё" необходимо сформировать новый шаблон из восьми чисел:

$$80,\ 0,\ 112,\ 136,\ 248,\ 128,\ 112,\ 0 \qquad (2)$$

и расположить их соответственно в ячейках ВП по адресам:

$$T,\ T+1,\ \ldots\ T+7\ (T = ASC(\text{"@"}) * 8 + \Delta).$$

Откуда взялась последовательность (2)? На схеме рисунка 9 указано правило получения шаблона (2) по изображению символа "ё" на сетке размера 8×8. Подобная схема применяется и во всех других случаях. При этом следует учитывать, что для экрана 0 используются лишь старшие 6 бит каждого байта. Следовательно, и рисовать символы необходимо на первых 6 колонках сетки. В противном случае выводимые знаки будут усекаться справа.

Указатели весов битов

байта	128	64	32	16	8	4	2	1	
T		●		●					→ 64+16 = 80
T+1									→ 0 = 0
T+2		●	●	●					→ 64+32+16 = 112
T+3	●				●				→ 128+8 = 136
T+4	●	●	●	●	●				→ 128+64+32+16+8 = 248
T+5	●								→ 128 = 128
T+6		●	●	●					→ 64+32+16 = 112
T+7									→ 0 = 0

Рис. 9

Следующий пример демонстрирует, как реально VROKE размещает компоненты шаблона для "ё" в видеопамяти. Далее, после выполнения строк 10—80 нажатие на клавишу @ приводит к индикации на экране буквы "ё". Восстановление истинного (своего) шаблона для знака @ реализуется командами SCREEN0 или SCREEN 1.

```
ПРИМЕР 64
10 SCREEN 0:D=2048:' или SCREEN 1:D=0
20 FOR J=0 TO 7
30 READ X:VPOKE ASC("@")*8+J+D,X
80 NEXT J
90 A$=INPUT$(1):PRINT A$:GOTO 90
100 DATA 80,0,112,136,248,128,112,0
```

VPEEK. Для функции VPEEK (t), где t — арифметическое выражение, значением является содержимое байта видеопамяти по адресу T=int(t) ($0 \leqslant T \leqslant 16383$).

STICK. Эта функция получает значения: 1, 3, 5 и 7 при нажатиях соответственно на клавиши: ↑, →, ↓, ←. Она может служить для программного распознавания типа нажимаемой клавиши управления курсором.


```
ПРИМЕР 65
10 X$=INKEY$:U=STICK(0)
20 IF U<>0 THEN PRINT U;
30 GOTO 10
```

Программная обработка ошибок. Кроме сбоев в работе компьютера из-за неисправностей каких-либо его узлов, имеется достаточно много причин, по которым происходит прерывание вычислений по программе и при нормально работающей машине. Перечислим некоторые из них:

а) для элемента массива получено значение индекса, не лежащее в диапазоне, указанном командой DIM;
б) при вычислениях получено число большее, чем допустимо;
в) в программе встретилась функция пользователя FN, которая не описана;
г) компьютеру предложено произвести деление некоторого числового значения на нуль.

Все эти и многие подобные им причины, вызывающие прерывания, имеют конкретные номера от 1 до 255, называемые кодами ошибок. Если какая-либо ошибка происходит, то на экране индицируется соответствующее сообщение. Например:

NEXT without FOR in 40
(NEXT без FOR в строке 40).

После устранения причин, вызвавших прерывание программы, как правило, приходится запускать ее заново. Однако имеется возможность «обработать» ошибку, не прекращая вычислений. Для этих целей используется команда

ON ERROR GOTO n (3)

где: ON (на), ERROR (ошибка), GOTO (идти) — служебные слова; n — номер программной строки.

При выполнении этой команды происходит назначение передачи управления на строку с номером n. Но сама передача реализуется лишь в случае возникновения ошибки.

Фрагмент программы, начинающийся со строки n и заканчивающийся командой

$$\text{RESUME}\left[\left\{\begin{matrix}0\\ \text{NEXT}\\ m\end{matrix}\right\}\right] \quad (4)$$

называется подпрограммой обработки ошибок, где: RESUME (возобновлять, продолжать), NEXT (следующий) — служебные слова; m — номер программной строки.

После обработки ошибки в зависимости от значения параметра, расположенного за словом RESUME, возврат в основную программу осуществляется либо к команде, вызвавшей ошибку (0 или нет параметров), либо к следующей за ней (NEXT), либо к строке с номером m.

Всегда действует последнее из выполненных назначений (3), а его отмена реализуется командой

ON ERROR GOTO 0

В подпрограммах обработки ошибок обычно используются значения функций ERR и ERL, равные соответственно коду последней ошибки и номеру строки, где она произошла.

Далее, командой ERROR α, где α — арифметическое выражение, можно искусственно вызывать ошибки с любыми кодами m = = INT (α) ($0 \leqslant m \leqslant 255$).

П Р И М Е Р 66

```
10 ON ERROR GOTO 90
20 E=10:Z=0
30 P=E/Z:PRINT "P=";P
40 Q=LOG(E-11)
50 ERROR 250
60 PRINT "ЗАВТРА":
70 ON ERROR GOTO 0:END
80 REM обработка ошибок
90 PRINT ERL,ERR
100 IF ERR=11 THEN 140
110 IF ERR=5 THEN 150
120 IF ERR=250 THEN 160
130 STOP
140 Z=.1:RESUME 0
150 RESUME NEXT
160 RESUME 60
```

```
run
 30                    11
P= 100
 40                    5
 50                    25w
ЗАВТРА
```

Переход к подпрограммам по событию. Имеется возможность организовать переходы на подпрограммы по совершению таких событий, как истечение определенного интервала времени, нажатие на клавиши: F1 — F10, CTRL/STOP, «пробел» и т. п. Об этом и пойдет речь в данном пункте.

Переход по временному интервалу. По команде

ON INTERVAL = t GOSUB n

где ON (по), INTERVAL (интервал), GOSUB (переход к подпрограмме) — служебные слова; t — целое из диапазона [0,65535], n — номер программной строки, назначаются интервалы времени, равные t/50 с, через которые периодически передается управление подпрограмме, расположенной со строки n. Отсчет временных интервалов реализуется специальным счетчиком, принимающим значения 1,2,...,t,1,2,...,t,1, ... Этот счетчик запускается (ON), сбрасывается (OFF) или приостанавливается (STOP) по команде

$$\text{INTERVAL} \left\{ \begin{matrix} \text{ON} \\ \text{OFF} \\ \text{STOP} \end{matrix} \right\}$$

П р и м е р. В этом фрагменте демонстрируется возможная организация учета времени выполнения программ, если оно больше одного цикла таймера.

П Р И М Е Р 67

```
10 T=0:TIME=0:H=60000
20 ON INTERVAL=H GOSUB 500
30 INTERVAL ON
   ...
90 PRINT "Время = ";(T+TIME)/50;"сек"
100 END
500 T=T+H:TIME=0
510 RETURN
```

Проследите за выполнением следующего примера при k = 1, 2, 6.

П Р И М Е Р 68

```
10 INPUT K
20 ON INTERVAL=K*50 GOSUB 80
30 INTERVAL ON:GOSUB 100
40 INTERVAL STOP:GOSUB 100
50 INTERVAL ON:GOSUB 100
60 INTERVAL OFF:GOSUB 100
70 END
80 PRINT "ПРОВЕРКА":RETURN
100 FOR K=1 TO 7000:NEXT:RETURN
```

Переход по ключу. По команде ON KEY GOSUB $n_1,n_2,...,n_k$, где: ON (по), KEY (ключ), GOSUB — служебные слова; n_1, n_2,...,n_k — номера строк программы ($k \leqslant 10$), могут осуществиться передачи управления подпрограммам, расположенным со строк n_1, n_2,...,n_k. При этом действие любого ключа FS (S=1,2, ..., k) должно быть или подтверждено (ON), или отменено (OFF), или приостановлено (STOP) командой

KEY(S) { ON / OFF / STOP }

Заметим, что если в момент действия параметра STOP для ключа FS произведено нажатие FS, то после KEY(S) ON сразу же реализуется соответствующая передача управления.

Прогон следующего фрагмента позволит более полно понять действие всех упомянутых выше команд.

```
П Р И М Е Р  69

10 KEY(1) ON:KEY(2) ON

20 ON KEY GOSUB 70,80

30 GOSUB 90

40 KEY(1) STOP:GOSUB 90

50 KEY(1) ON:GOSUB 90:END

60 REM подпрограммы

70 PRINT "НАЖАТА КЛАВИША F1":RETURN

80 PRINT "НАЖАТА КЛАВИША F2"

85 KEY(2) OFF:RETURN

90 FOR K=1 TO 2000:NEXT:RETURN
```

Переход по CTRL — STOP. По команде ON STOP GOSUB n, где: ON, STOP, GOSUB — служебные слова; n — номер программной строки, при нажатии на клавиши CTRL — STOP может осуществиться передача управления подпрограмме, расположенной со строки n. При этом действие CTRL — STOP должно быть подтверждено (ON), отменено (OFF) или приостановлено (STOP) командой

STOP { ON / OFF / STOP }

Переход по пробелу. По команде ON STRIG (0) GOSUB n, где: ON, STRIG — служебные слова; n — номер программной строки,

при нажатии на клавишу «пробел» может осуществиться передача управления подпрограмме, расположенной со строки n. При этом действие клавиши «пробел» должно быть подтверждено (ON), отменено (OFF) или приостановлено (STOP) командой

STRIG(0) { ON / OFF / STOP }

```
П Р И М Е Р  70

360 ON STRIG GOSUB 440

370 STRIG(0) ON:GOSUB 430

380 STRIG(0) STOP:GOSUB 430

390 STRIG(0) ON:GOSUB 430

400 STRIG(0) OFF:GOSUB 430

410 GOSUB 430:END

430 FOR K=1 TO 1000:NEXT K:RETURN

440 PRINT "ПРЕРЫВАНИЕ ПО STRIG":RETURN
```

Отладка программ. Написав программу, прежде всего необходимо проверить ее на данных, по возможности охватывающих все случаи, которые могут встретиться при реальном счете. Для этого составляются и «прокручиваются» различные тесты. Они создаются с учетом специфики решаемых задач, выбранного алгоритма и степени риска от возможных необнаруженных ошибок. Довольно часто это просто серия специально подобранных контрольных задач, не требующих значительных затрат времени на решение, в которых заранее известны ключевые, а во многих случаях и промежуточные результаты вычислений. Их сопоставление со значениями, полученными при счете, позволяет высказать более или менее правдоподобные гипотезы о правильности программы. Если в ней обнаружится ошибка, а это почти всегда так, то процесс тестирования должен быть приостановлен. Далее начинается процесс отладки, т. е. выявление типа ошибок, их локализация и исправление. Успешному проведению отладки способствует выдача различных сообщений о ходе решения конкретных задач. В частности, начиная с любой команды, можно проследить «трассу» дальнейшего хода вычислений. Для этого в требуемом месте размещают команду TRON.

Ее реализация приводит к включению режима трассировки, при котором последовательно выводятся номера выполняемых в данный момент строк в виде: [n_1] [n_2] ... [n_k]...

Между компонентами [ ] могут располагаться значения, печатаемые по командам PRINT или вводимые с клавиатуры по командам INPUT.

```
ПРИМЕР 71

10 TRON                    run
20 A=5:B=6:PRINT A,B       [20] 5          6
30 IF A*B<99 THEN 100      [30][100] 11
110 PRINT B-A              [110] 1
120 TROFF                  [120]
```

Анализ полученной трассы, как правило, позволяет локализовать ошибку.

Отменяется режим трассировки командой TROFF.

Обе команды TRON и TROFF можно выполнять в непосредственном счете.

Другой, часто применяемый способ отладки, состоит в следующем. В «критических» точках программы расставляются вспомогательные команды STOP.

При выполнении STOP вычисления приостанавливаются и пользователь в непосредственном счете может вывести значения интересующих его переменных. Анализ их позволяет делать выводы о правильности хода вычислительного процесса и соответственно принимать те или иные решения.

После останова по команде STOP вычисления возобновляются командой CONT, которая должна быть выполнена в непосредственном счете.

2. ГРАФИКА

Айвен Сазерленд — пионер применения компьютеров для построения и обработки изображений — заметил как-то о своем изобретении: «Дисплей, подключенный к ЭВМ, представляется мне окном в Алисину Страну чудес, где программист может изображать любые объекты, описываемые хорошо известными законами природы, либо чисто воображаемые объекты, подчиняющиеся законам, записанным в программе. С помощью дисплеев я сажал самолет на палубу движущегося авианосца, следил за движением элементарной частицы в потенциальной яме, летал в ракете с околосветовой скоростью и наблюдал за таинствами внутренней жизни вычислительной машины».

Восторг доктора философии понять несложно. Возможности машинной графики воистину безграничны. Особенно впечатляющим представляется ее применение в системе образования при разработке и использовании педагогических программных средств. Простота

освоения элементов машинной графики способствует ее проникновению в разнообразные прикладные сферы и, несомненно, обеспечивает дальнейший рост ее популярности.

2.1. Экраны. Пикселы. Точки

Дисплей ЭВМ «Ямаха» имеет два режима для вывода графической информации. Их называют соответственно SCREEN2 и SCREEN3 (*screen* — экран). Закодированное изображение хранится в специальном участке памяти, называемом *буфером образа,* где оно представлено в форме матрицы двоичных значений интенсивностей свечения или цветов точек экрана. Блок сопряжения, именуемый *дисплейным контроллером,* обрабатывает содержимое буфера образа и преобразует двоичный код в видеосигналы, передаваемые дальше в *телевизионный монитор.* Возникающая в результате этого на экране картина воспринимается пользователем как устоявшаяся потому, что она возобновляется с частотой, не меньшей 30 раз в секунду.

Наименьшим неделимым элементом изображения, который может быть показан на экране, является маленькое светящееся пятно, именуемое пикселем. Общее количество пикселей экрана (режима) является важной его характеристикой.

Наряду с пикселями введем понятие точки на экране. Пусть задана декартова прямоугольная система координат так, как это показано на рисунке 10. Обратите внимание на непривычное расположение начала координат и направление оси ординат. Впрочем, как показывает опыт, приспособиться к этой системе довольно просто.

Множества координатных точек (a, b) и их общее количество для обоих экранов одинаковы:

$$(a,\ b)\quad \begin{matrix} a \in \{0,\ 1,\ \ldots\ 255\} \\ b \in \{0,\ 1,\ \ldots\ 191\} \end{matrix}$$

$$P = 256 \times 192 = 49152$$

Однако допустимое адресуемое пространство значительно шире. Некоторые параметры графических команд разрешается задавать в диапазоне от —32768 до 32767.

Пикселы для обоих экранов представляют собой маленькие квадратики, задаваемые координатами верхнего левого угла. Каждая координатная точка определяет по одному пикселю. Но размеры их для экранов 2 (слева) и 3 (справа) разные. Из рисунка 11 можно

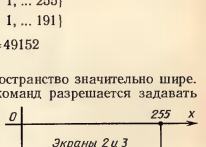


Рис. 10

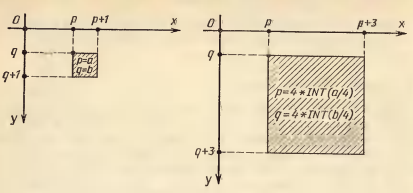


Рис. 11

понять, как точка (a, b) задает пиксель и каковы его размеры для обоих экранов.

Заметим, что один пиксель экрана 3 соответствует блоку из 4×4 пикселей экрана 2. Далее, нетрудно видеть, что для экрана 2 мы имеем взаимно однозначное соответствие между множествами координатных точек и пикселями.

В любой конкретный момент времени одна из точек (a, b) экрана находится в особом состоянии, или, как говорят, отмечена графическим курсором. *a* и *b* в этом случае называют текущими координатами курсора. При включении компьютера графический курсор устанавливается в положение (0, 0) — верхний левый угол экрана.

Пусть (a, b) — текущая позиция курсора, x и y — арифметические выражения, у которых используются целые части x_0 и y_0 соответствующих значений.

В описании графических команд будут встречаться следующие элементы:

1. @ — необязательный элемент команд никоим образом не влияющий на их выполнение. Этот символ фактически является лишь «фирменным» знаком совокупности графических команд;

2. (X, Y) без параметра STEP — необязательный элемент, определяющий точку экрана (x_0, y_0). Если (x_0, y_0) вне пределов экрана, то элемент, как правило, не используется или вызывает прерывание при выполнении соответствующей команды;

3. STEP (X, Y) — элемент, задающий смещение относительно текущей позиции курсора (a, b) и определяющий точку (a+x_0, b+y_0). Если эта точка лежит вне пределов экрана, то элемент, как правило, не используется или вызывает прерывание при выполнении соответствующей команды;

4. C (или C_1) — арифметическое выражение, целая часть значения которого лежит в диапазоне [0, 15] и определяет цвет точки, области и т. п. По умолчанию C есть цвет изображения.

Многие элементы в графических командах не являются обязательными и могут быть опущены. Однако соответствующая запятая в тексте должна сохраниться и тем самым указывать на отсутствие того или иного параметра. Это правило не касается завершающих запятых. Их оставлять нельзя.

В последующих пунктах дается описание каждой из системы базовых графических команд и функций Бейсика. С их помощью мы научимся выводить на экран точки, линии, прямоугольники, окружности, эллипсы и дуги, покажем, как заполнять области экрана однотонным цветом, и рассмотрим простейшие примеры использования этих средств.

2.2. Установка и стирание точек

В этом пункте речь пойдет о двух командах PSET и PRESET, используемых для перемещения графического курсора в требуемую точку с возможным изменением цвета соответствующего ей пикселя, а также функции POINT, позволяющей узнавать цвет конкретных пикселей.

. *Команда* PSET(*установить точку*). Записывается команда в виде

PSET [@] [STEP] (X, Y) [,C]

Выполняется PSET так. По элементам (X, Y) или STEP (X, Y) находится точка (x_0, y_0), а по ней — левый верхний угол (p, q) соответствующего пикселя. Графический курсор перемещается в точку (p, q) и раскрашивает пиксель в цвет C. Следует иметь в виду, что для экрана 2 это может привести к изменению цвета до 8 близлежащих пикселей, расположенных в строке p.

ПРИМЕР 1. Точки (пиксели) с разной раскраской

```
10 COLOR 15,1,8:SCREEN 2
20 PSET (7,3)
30 FOR X=0 TO 15
40 PSET STEP(17,13),X
50 NEXT X
60 GOTO 60
```

ПРИМЕР 2. Синусоида

```
10 COLOR 1,15,8:SCREEN 2
20 PSET (0,95)
30 FOR K=0 TO 255:PSET STEP(1,0),1:NEXT
40 PSET (128,0)
   FOR K=0 TO 191:PSET STEP(0,1),1:NEXT
   Z=8*ATN(1):'------------------- 2*пи
   FOR K=0 TO 255
   PSET (K,50*SIN(Z*K/256)+95),5
   NEXT
1   GOTO 100
```

Команда PRESET (*стереть точку*). Записывается команда в виде

PRESET [@] [STEP] (X, Y) [,C]

При выполнении команды PRESET без параметра C соответствующий ей пиксель «стирается», т. е. закрашивается под цвет фона. Если параметр C присутствует, то действия команд PRESET и PSET идентичны.

ПРИМЕР 3. Установка и стирание точек (бегущий отрезок)

```
10 COLOR 1,15,8:SCREEN 2
20 FOR K=10 TO 265
30 PSET (K,95),X:PRESET(K-10,95)
40 NEXT K:GOTO 20
```

Функция POINT (точка). Записывается функция POINT в виде

POINT (X, Y)

Ее значением является код цвета соответствующего ей пикселя или —1, когда координаты точки выходят за пределы экрана.

ПРИМЕР 4. Случайное задание цвета пикселей

Поиск среди них голубых пикселей

```
10 COLOR 1,15,8:SCREEN 3
20 FOR K=1 TO 12
30 PSET (20*K,95),INT(RND(-TIME)*16)
40 NEXT K:P=1
50 FOR K=1 TO 12
60 IF POINT (20*K,95)=7 THEN P=2
70 NEXT K
80 FOR K=1 TO 500:NEXT K
90 SCREEN 0
100 ON P GOSUB 120,130
110 END:'----------------подпрограммы
120 PRINT "голубых точек нет":RETURN
130 PRINT "голубые точки есть":RETURN
```

2.3. Круги. Эллипсы. Дуги

По команде CIRCLE, записываемой в виде

CIRCLE [@] [STEP] (X, Y), R, [C], [α], [β], [e] (1

вычерчиваются окружности (круги), эллипсы или их дуги и секторы. Смысл параметров в (1) следующий.

1. CIRCLE (круг, окружность) — служебное слово.
2. (X, Y) или STEP (X, Y) — параметры, определяющие абсолютные или относительные координаты центра окружности и новое место — положение курсора.
3. R — арифметическое выражение. INT(R) задает радиус окружности.
4. α, β — арифметические выражения, фиксирующие углы в радианах, в пределах которых рисуется дуга окружности. По умолчанию $\alpha = 0$, $\beta = 2\pi$ ($\approx 6{,}28318$)
5. e — арифметическое выражение, задающее коэффициент сжатия окружности. Значение e должно лежать в пределах $1/260 \leqslant e \leqslant 260$. При $e<1$ происходит сжатие вдоль оси Oy, а при $e>1$ — вдоль оси Ox. По умолчанию $e=1$

Как находятся координаты центра окружности? Если он определяется элементом (X, Y), то берутся целые части значений соответствующих выражений:

$$X_0 = INT(X),\ Y_0 = INT(Y)$$

и по точке (x_0, y_0) отыскивается левый верхний угол (p, q) ее пикселя. Это и есть центр O окружности и новая позиция курсора после выполнения команды. Если центр O определяется элементом STEP (X, Y), то в нахождении (p, q) участвуют текущие координаты (a, b) курсора. А именно,

$$X_0 = INT(X) + a,\ Y_0 = INT(Y) + b$$

и далее все так же, как и в предыдущем случае.

Напомним, что для экрана 2 $(x_0, y_0) = (p, q)$.

Несмотря на необычное расположение осей координат, отсчет углов вполне традиционен. Он ведется от оси Ox против часовой стрелки. Поскольку углы измеряются в радианах, то в некоторых случаях для задания их значений удобно использовать стандартную функцию atn(x) (arctg(x)), для которой

$$atn(1) = \pi/4(45°)$$

ПРИМЕР 5. Смещенные фигуры

```
10 SCREEN 0:INPUT "вводите 1, 2 или 3 ";X
20 COLOR 1,15,8:SCREEN 2
30 PSET (0,85)
40 FOR K=1 TO 50
50 ON X GOSUB 80,90,100
60 NEXT K
70 GOTO 70:'----------------подпрограммы
80 CIRCLE STEP(5,0),50,4:RETURN
90 CIRCLE STEP(5,0),50,4,0,4*ATN(1):RETURN
100 CIRCLE STEP(5,0),50,4,,,,5:RETURN
```

По данному фрагменту индицируются 50 одинаковых фигур со смещениями. Тип выводимой фигуры зависит от ответа пользователя на запрос с экрана дисплея. При x=1, 2 и 3 получаем соответственно окружности, полуокружности и эллипсы. Проанализируйте команды CIRCLE в подпрограммах 80, 90 и 100, реализующие этот вывод.

ПРИМЕР 6. 'Арбуз'

```
10 COLOR 1,15,8:SCREEN 2
20 FOR K=1 TO 100
30 C=INT(RND(1)):E=RND(1)
40 CIRCLE (128,95),80,C,,,E
50 NEXT K
60 GOTO 60
```

В этом примере вычерчивается ряд цветных эллипсов с совпадающей большей осью. На монохромном дисплее получается фигура, напоминающая арбуз.

Команду CIRCLE можно также использовать для вычерчивания секторов окружностей и эллипсов. Для этого необходимо задать отрицательные значения параметрам α и β. При индикации соответствующей дуги будут взяты $|\alpha|$ и $|\beta|$, но концы дуги соединятся отрезками с центром. Нуль для этого случая надо задавать малым отрицательным числом, например —0,001.

ПРИМЕР 7. Сектор окружности

```
10 COLOR 1,15,8:SCREEN 2
20 CIRCLE (100,95),60,C,-1E-03,-2*ATN(1)
30 GOTO 30
```

2.4. Отрезки прямых. Прямоугольники

По команде LINE, записываемой в виде:

LINE [[@] [STEP] (X_1, Y_1)] — [@] [STEP] (X_2, Y_2), [C], { [B] / [BF] }

вычерчиваются отрезки прямых и прямоугольники с одновременной раскраской их в цвет. Смысл параметров здесь следующий.

LINE (линия) — служебное слово;

B — параметр, определяющий рисование прямоугольника цветом C без закраски его внутренней части;

BF — параметр, определяющий рисование прямоугольника с одновременной закраской его цветом C.

Элементы (x_i, y_i) или STEP (x_i, y_i) $(i = 1, 2)$ задают на экране две точки M_1 и M_2. Находятся M_1 и M_2 тем же способом, каким отыскивался центр O окружности в предыдущем пункте. Если первый из упомянутых элементов отсутствует, то в качестве M_1 берется текущая позиция (a, b) графического курсора. Результирующей позицией курсора является точка M_2.

На выполнение команды LINE существенное влияние оказывают параметры B и BF или их отсутствие. Рассмотрим три случая.

I. B и BF отсутствуют. Тогда на экране цветом C проводится линия, соединяющая точки M_1 и M_2.

2. Есть параметр B. Тогда цветом C со сторонами, параллельными осям координат, рисуется прямоугольник. Точки M_1 и M_2 оказываются его противоположными вершинами. Внутренняя часть прямоугольника не закрашивается.

3. Есть параметр BF. Тогда так же, как и в случае 2, рисуется прямоугольник, но внутренняя часть его закрашивается в цвет C.

```
П Р И М Е Р  8. Возможные фигуры
10 COLOR 1,15,8:screen 2
20 '----------------------отрезок
21 LINE (5,5)-(25,25)
30 '-----------------прямоугольник
31 LINE -(70,60),,B
40 '-----------------прямоугольник
41 LINE STEP(8,8)-STEP(130,70),,B
50 '-----закрашенный прямоугольник
51 LINE STEP(10,10)-(90,80),,BF
60 GOTO 60
П Р И М Е Р  9. Смещенные или вложенные
                прямоугольники
10 SCREEN 0:INPUT"вводите 1 или 2 ";R
15 COLOR 1,15,12:SCREEN 2
20 FOR X=0 TO 15
25 A5=5*X:A6=A5+X:A7=A6+X:A8=A7+X
30 ON R GOSUB 45,60
35 NEXT:FOR S=1 TO 800:NEXT:GOTO 10
40 '-----------------смещенные прямоугольники
45 LINE (40+A7,30+A5)-(130+A7,90+A5),15-X,BF
50 RETURN
55 '-----------------вложенные прямоугольники
60 LINE (5+A8,10+A6)-(250-A8,185-A6),15-X,BF
65 RETURN
```


По данному фрагменту в зависимости от ответов пользователя на запрос с экрана дисплея индицируются 16 смещенных по отношению друг к другу или 16 вложенных разноцветных прямоугольников.

В следующем примере демонстрируется возможность использования в команде PSET и элементе STEP (X, Y) отрицательных значений.

```
П Р И М Е Р  10. Отрицательные значения
10 COLOR 1,15,8:SCREEN 3:PSET(-4,160)
20 FOR X=0 TO 15
30 LINE STEP(8,-130)-STEP(8,130),X,BF
40 NEXT X
50 GOTO 50
П Р И М Е Р  11. Звездное небо
10 COLOR 7,1,8:SCREEN 2
20 DEFFNS(Z)=INT(RND(1)*Z)
30 X=FNS(256):Y=FNS(192)
40 E=FNS(16):H=FNS(2)
50 LINE (X,Y)-STEP(H,H),E,BF
60 FOR K=1 TO 40:NEXT K:GOTO 30
```

2.5. Закрашивание областей

Пусть G — некоторая, не обязательно односвязная область экрана и L — ее граница. Последняя может состоять из одной или нескольких замкнутых линий (см. рис. 12).

Для закрашивания области G используется команда PAINT (красить). Сразу же оговоримся, что для экрана 2 возможна раскраска G только под цвет границы L. Для экрана 3 это ограничение не действует. Внутренность G можно раскрасить любым цветом.

Записывается команда PAINT так:

PAINT [@] [STEP] (X, Y), [C], [C_1] (3)

Рис. 12

где: (X, Y) или STEP (X, Y) — элемент, определяющий координаты (p, q) пикселя внутри G (см., например, пункт 2.3); C — арифметическое выражение, задающее цвет закраски области G; C_1 — арифметическое выражение, задающее цвет границы L (для экрана 3!).

Опишем выполнение команды (3) отдельно для разных экранов.

SCREEN 2. Область G раскрашивается в цвет C, совпадающий с цветом границы L. По умолчанию C есть цвет изображения.

SCREEN 3. Область G раскрашивается в цвет C. По умолчанию C есть цвет изображения. Если раскраска G производится не под цвет L, то присутствие параметра C_1 обязательно.

Подчеркнем, что во всех случаях точка (p, q) должна быть внутри G (не на границе !).

В примере 12 производится раскраска G под цвет, отличный от L. В примере 13 под цвет L раскрашиваются вложенные эллипсы.

ПРИМЕР 12. Раскраска многосвязной области

```
10 COLOR 1,15:SCREEN 3
20 LINE (30,30)-(225,162),,B:'прямоугольник
30 CIRCLE (80,95),30          :'окружность
40 CIRCLE (175,95),30,,,,.5 :'эллипс
50 PAINT (125,95),7,1
60 GOTO 60
```

ПРИМЕР 13. Вложенные эллипсы

```
10 COLOR 1,15:SCREEN 2
20 FOR K=0 TO 15
30 CIRCLE (128,95),127-8*K,15-K,,,.7
40 PAINT (128,95),15-K
50 NEXT K
60 GOTO 60
```

Возникает вопрос: можно ли вообще раскрасить на экране 2 область G под цвет, отличный от контура? Для некоторых прямоугольников и других областей, границы которых состоят только из отрезков, параллельных осям координат, это сделать удается. Иллюстрацией тому является следующий фрагмент.

ПРИМЕР 14. Экзотическая раскраска

```
10 COLOR 1,15:SCREEN 2
20 LINE (79,79)-(120,120),,B
30 LINE (80,80)-(119,119),7,B
50 PAINT (100,100),7
60 GOTO 60
```

Здесь команда 20 выводит прямоугольник P, а команды 30 и 50 раскрашивают его в цвет 7. Наложения цветов не происходит из-за удачного расположения на экране вершин P. Вообще, если P определен точками (X_1, Y_1) и (X_2, Y_2), где:

$$X_1 \text{MOD } 8 = 7 \quad \text{и} \quad X_2 \text{MOD } 8 = 0$$
$$Y_1 \text{MOD } 8 = 7 \qquad\quad Y_2 \text{MOD } 8 = 0,$$

то указанным приемом его всегда можно раскрасить в любой цвет без каких-либо искажений.

2.6. Язык GML

Для расширения возможностей, предоставляемых ранее разобранными средствами машинной графики, дополнительно разработан специальный макроязык GML (*Graphics Macro Language*), позволяющий в компактной форме кодировать любые фигуры и быстро выводить их на экран в требуемом масштабе и различных ориентациях. Каждая команда GML представляется одиночной ключевой буквой, за которой следует один или два числовых параметра. Последние, как правило, задаются целыми числами. В основном команды GML (см. табл. 12) предназначены для перемещения курсора из одних точек экрана в другие с одновременным вычерчиванием соединяющих их отрезков. Впрочем, они могут модифицироваться следующим образом.

1. Перед командой «движения» ставится параметр B. Тогда положение курсора изменяется, но линия не рисуется.

2. Перед командой «движения» ставится параметр N. Тогда курсор возвращается на место, которое он занимал до выполнения данной команды.

Там, где в команде GML в качестве параметров используются целые числа, можно указывать и числовые переменные v, в том числе и элементы массивов. Но v при этом следует записывать в форме =v;.

Примеры команд GML:

M 50, 60	BM 30, 40	NM 20, 10
R 10	C 7	M=X;,=Y;
U 20	C 15	C=AB;
E 30	S 8	U=A(3);
L 40	S 24	M+=C;,−=D;
M −5, +15	A 2	H=Z;

Таблица 12

Команды языка GML

Команда	Название и выполняемые действия
	Перемещение курсора по направлению v. Здесь: 1. v — указатель одного из 8 возможных направлений (см схему):  v∈{R, E, U, H, L, G, D, F} Углы между соседними направлениями равны 45°.
1 2 ... vn	
8	2. *n* — целое число, задающее величину смещения курсора в точках экрана. При выполнении команды курсор смещается от своего текущего положения на *n* единиц по направлению v
9 MX Y	Перемещение курсора из текущего положения в точку с координатами (X, Y)
10 M±X,±Y	Смещение курсора по отношению к текущей позиции на ±X единиц по абсциссе и ±Y единиц по ординате. На относительность смещения указывают знаки "+" или "—" у X, которые проставлять обязательно. Знак "+" у Y необязателен
11 Cp	Определение цвета вычерчиваемых фигур (p=0,15 — код цвета). По умолчанию p есть цвет изображения. Действует Cp во всех командах DRAW до нового назначения
12 Sm	Задание масштабного множителя m (0≤m≤255). При выполнении Sm все числовые параметры всех команд движения, записанных без префикса B, получают масштабный множитель m/4. Действует Sm во всех командах DRAW до нового назначения. Заметим, что S0=S4
13 At	Поворот изображения вокруг точки, с которой начиналось рисование, на 90°×*t* (*t*∈{0, 1, 2, 3}) против часовой стрелки. Действует At во всех командах DRAW до нового назначения.
14 Xα;	Интерпретация значения строковой переменной α как последовательности команд GML. Знак ";" обязателен.

Для приведения в действие последовательности команд GML необходимо сделать ее значением некоторого строкового выражения β и затем воспользоваться командой DRAW (чертить, рисовать):

DRAW β

Как правило, β является константой. Выполнение этой команды состоит в последовательном выделении из значения β команд GML и реализации предписываемых ими операций. Длина β не должна превосходить 255 байтов. Однако имеется довольно простой механизм «вложения» одних значений в другие. Фактически в β можно вставлять команду обращения к другой серии команд GML. Записывается она так: Xα;. Знак ; обязателен. Выполнение этой команды активизирует подпрограмму на GML, являющуюся значением строкового выражения α. Возврат в β осуществляется к команде, следующей за Xα;. Допускаются многоуровневые обращения типа Xα;, т. е. из α можно обратиться к некоторой третьей подпрограмме и т. д.

Заметим, что если не предполагается масштабировать выводимую фигуру или поворачивать ее вокруг начальной точки, то все равно есть смысл указать в DRAW команды S4A0. Это позволит отменить возможные предыдущие назначения Sm и At, могущие исказить ваше изображение. Далее, чтобы текст на GML легко читался и корректировался, рекомендуется отделять его команды в β пробелами. На выполнении команды DRAW это никак не отразится.

П Р И М Е Р 15. Правильный шестиугольник

```
10 COLOR 15,1,8:SCREEN 2
20 DRAW "BM70,15 S4 A0"
30 FOR K=1 TO 6:READ X,Y
40 DRAW "M+=X;,=Y;"
50 NEXT K
60 GOTO 60
70 DATA 45,0,25,40,-25,40
80 DATA -45,0,-25,-40,25,-40
```

Здесь присутствуют следующие команды GML:

BM 70,15	— перемещение курсора в точку (70,15) без рисования;
S4	— задание единичного масштаба;
A0	— задание угла поворота в 0°;
M+ =X;,=Y;	— рисование сторон шестиугольника. X и Y в M — числовые переменные.

П Р И М Е Р 16. Правильные шестиугольники. Масштабирование

```
10 COLOR 15,1,8:SCREEN 2
20 FOR Z=1 TO 9
30 DRAW "BM70,10 S=Z; A0"
40 DRAW "M+45,0 M+25,40 M-25,40"
50 DRAW "M-45,0 M-25,-40 M+25,-40"
60 NEXT Z
70 GOTO 70
```

В этом примере демонстрируется другой способ рисования правильного шестиугольника. Команда S=Z; определяет масштаб, равный значению Z.

П Р И М Е Р 17. Правильные шестиугольники. Повороты

```
10 COLOR 15,1,8:SCREEN 2
15 FOR U=0 TO 3
20 FOR Z=1 TO 5
30 DRAW "BM128,95 S=Z; A=U;"
40 DRAW "M+45,0 M+25,40 M-25,40"
50 DRAW "M-45,0 M-25,-40 M+25,-40"
60 NEXT Z,U
70 GOTO 70
```

П Р И М Е Р 18. Орнамент. Перемещение по направлениям

```
10 COLOR 1,15,8:SCREEN 2
20 FOR J=20 TO 160 STEP 20
30 FOR I=20 TO 200 STEP 20
40 DRAW"S4 A0 BM=I;,=J;"
50 DRAW"E10 R10 F10 D10 G10 L10 H10 U10"
60 NEXT I,J
70 GOTO 70
```


2.7. Программа «Базовые структуры»

В качестве примера применения команд GML приведем программу «Базовые структуры», по которой организуется рисование основных структур действий, используемых при описании алгоритмов (см. рис. 13—15). Вывод конкретной схемы алгоритма организуется нажатием на клавиши F1 — F5. Завершается работа с программой при нажатии на клавишу F6.

```
10 '    Б А З О В Ы Е   С Т Р У К Т У Р Ы
20 '-----------------------------------------
30 COLOR 1,15,12:SCREEN 2
40 'T1$ - T3$  -  стрелки
50 'T4$  -  прямоугольник, T5$  -  ромб
60 T1$="NH3 NG3":T2$="NG3 NF3":T3$="NH3 NE3"
70 T4$="M+0,14 M+56,0 M+0,-28 M-56,0 "
80 T4$=T4$+"M+0,28 BM+56,-14"
90 T5$="E14 F14 G14 H14 E14":DRAW "A0 S4"
100 OPEN "GRP:"FOROUTPUT AS#1
110 FOR K=1 TO 6:KEY(K) ON:NEXT
120 ON KEY GOSUB 140,230,280,500,630,750
130 GOTO 120
140 '-----------------------------------------
150 CLS
160 DRAW "BM10,75M+42,0XT1$;XT4$;M+42,0XT1$;"
170 DRAW "XT4$;M+42,0 XT1$;BM-170,-3"
180 PRINT #1,"A";
190 DRAW "BM+90,0":PRINT#1,"B":DRAW"BM10,140"
200 PRINT #1,"структура 'ПОСЛЕДОВАТЕЛЬНОСТЬ'"
210 DRAW "BM100,160":PRINT#1,"A  :  B":RETURN
220 '-----------------------------------------
230 GOSUB 350:DRAW "M+56,0 BM5,140"
240 PRINT#1,"структура 'ВЕТВЛЕНИЕ'(неполное)"
250 DRAW "BM70,160":PRINT #1,"if  P  then  A"
```

```
260 RETURN
270 '-----------------------------------------
280 GOSUB 350:DRAW "XT1$; XT4$; BM-31,-3"
290 PRINT #1,"B";:DRAW "BM15,140"
300 PRINT #1,"структура 'ВЕТВЛЕНИЕ' (полное)"
310 DRAW "BM33,160"
320 PRINT #1,"if  P  then  A  else  B"
330 RETURN
340 '-----------------------------------------
350 GOSUB 460
360 DRAW "XT5$; M+0,-14 M+28,0 XT1$; XT4$;"
370 DRAW "M+28,0 M+0,22 XT3$;"
380 CIRCLE STEP(0,6),5
390 DRAW "BM+6,0 M+28,0 XT1$; BM-34,6"
400 DRAW "XT2$; M+0,22 M-28,0 BM-58,0"
410 DRAW "M-26,0 M+0,-14 BM-2,-17"
420 PRINT#1,"P";:DRAW "BM-2,-16":PRINT#1,"+";
430 DRAW "BM-8,32":PRINT #1,"-";
440 DRAW "BM+41,-45":PRINT #1,"A";
450 DRAW "BM-34,60":RETURN
460 CLS:DRAW "BM20,75 M+28,0 XT1$;":RETURN
470 GOSUB 460:CIRCLE STEP(6,0),5
480 DRAW "BM+5,0 M+28,0 XT1$;":RETURN
490 '-----------------------------------------
500 GOSUB 470:DRAW "XT5$; F14 M+28,0 XT1$;"
510 DRAW "XT4$; M+14,0 M+0,-42 M-159,0"
520 DRAW "M+0,36 XT3$; BM+47,21 M+0,28"
530 DRAW "M+130,0 XT1$; BM-132,-46"
540 PRINT #1,"P";:DRAW "BM+9,-5":PRINT #1,"+";
550 DRAW "BM-20,22":PRINT #1,"-";
```

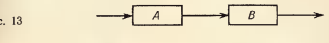


```
560 DRAW "BM+57,-17":PRINT #1,"A";
570 DRAW "BM30,140"
580 PRINT #1,"структура 'ЦИКЛ' (пока)"
590 DRAW "BM20,160"
600 PRINT #1,"μ  if  P  then  A : GOTO  μ"
610 RETURN
620 '-----------------------------------------
630 GOSUB 470
640 DRAW "XT4$;M+28,0 XT1$; XT5$; F14 M+28,0"
650 DRAW "XT1$; BM-42,-14 M+0,-28 M-131,0"
660 DRAW "M+0,36 XT3$; BM+59,3":PRINT #1,"A";
670 DRAW "BM+62,0":PRINT #1, "P";
680 DRAW "BM-2,-15":PRINT #1,"-";
690 DRAW "BM+4,20":PRINT #1,"+";
700 DRAW "BM40,140"
710 PRINT #1,"структура 'ЦИКЛ' (до)"
720 DRAW "BM36,160"
730 PRINT #1,"μ  A : if  ¬P  then  μ"
740 RETURN
750 END:'--имя программы на диске "СТРУКТУРЫ"
```

На рисунке 13 изображена базовая структура ПОСЛЕДОВАТЕЛЬНОСТЬ.

На рисунке 14 изображена неполная базовая структура ВЕТВЛЕНИЕ, а на рисунке 15 — полная.

Рис. 13



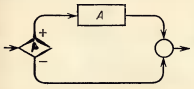


Рис. 14

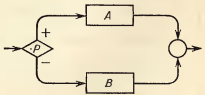


Рис. 15

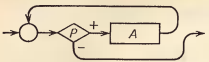


Рис. 16

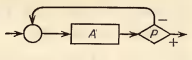


Рис. 17

На рисунке 16 изображена базовая структура ЦИКЛ-ПОКА, на рисунке 17 ЦИКЛ-ДО.

Схемы, графики, рисунки или отдельные их фрагменты, выводимые на графические экраны, обычно нуждаются в обозначениях и поясняющих сообщениях. Сформировать их обычным оператором PRINT невозможно. На экранах 2 и 3 последний не действует. Можно было бы буквы рисовать командой DRAW, но это весьма трудоемко. Надписи в программе «базовые структуры» формировались так, как это демонстрирует следующий фрагмент.

П Р И М Е Р 20. Символы на графическом экране (1)

```
10 COLOR 1,15,8:SCREEN 2
20 OPEN "GRP:"AS #1          40 PRINT #1,"пример"
30 PSET (100,100),15         50 GOTO 50
```

Этот способ позволяет выводить символы фиксированного формата в блоках 8×8 точек. Другой прием вывода символов алфавита на графический экран, позволяющий регулировать как размер букв, так и расстояние между ними, приводится в следующем пункте.

Разработка программы «Базовые структуры» является простейшим педагогическим программным средством, которое с успехом может быть использовано преподавателем при объяснении соответствующей темы по информатике. Программная реализация подобных разработок несложна, но требует значительных временных затрат. Связано это с недостаточностью инструментальных средств Бейсика для эффективного проектирования изображения непосредственно на экране дисплея. Стоит заметить, что многие школьники и студенты в качестве первой попытки создать «большую» и интересную программу с удовольствием и небезуспешно берутся за разработку и реализацию той или иной версии «Графического редактора», позволяющего облегчить проектирование информационных кадров для обучающих систем.

2.8. Алфавит на графическом экране

Пусть α — стандартный алфавит Бейсика. Один из способов оформления графиков и чертежей надписями из символов α демонстрировался в предыдущем пункте. При необходимости работать с символами различных форматов или с наборами знаков,

не совпадающих с подмножествами α, используется другой прием. Суть его состоит в том, что заранее заготавливаются строковые константы. Их значениями являются тексты на GML «новых» знаков, которые могут быть любыми фрагментами произвольных изображений. Далее, каким-либо способом «новые» знаки кодируются элементами α и выводятся на экран заданием соответствующего кода. Описанный прием реализуется в примере 21.

П Р И М Е Р 21. Символы на графическом экране (2)

```
10 '        СИМВОЛЫ НА ГРАФИЧЕСКОМ ЭКРАНЕ
20 '-------------------------------------------
30 ' (X,Y) - начальная позиция
40 ' B     - размер символов
50 ' L     - расстояние между символами
60 COLOR 1,15,12:SCREEN 2:CLEAR 1000
70 '---------------------------------------символы
80 DIM A$(255):A$(235)="D6U3R1E3G3F3 BM+0,-6"
90 A$(236)="BM+0,6 E1 U4 E1 R2 D6 BM+0,-6"
100 A$(225)="BM+0,6U4E2F2D2 L4 R4 D2 BM+0,-6"
110 A$(243)="BM+4,1H1L2G1D4 F1 R2 E1 BM+0,-5"
120 A$(53)="BM+20,4 NU20 NE20 NR20 NF20 XR$;"
130 R$="ND20 NG20 NL20 NH20"
140 '-------------------------------------------
150 TE$="КЛАСС5":X=30:Y=80:B=8:L=7
160 DRAW "BM=X;,=Y; A0 S=B;"
170 FOR K=1 TO LEN(TE$)
180 Z$=MID$(STR$(ASC(MID$(TE$,K,1))),2)
190 P$=P$+"XA$("+Z$+");"+"BR=L;"
200 NEXT:DRAW "XP$;"
210 GOTO 210:'-----------------------------------
```

Пример носит чисто иллюстративный характер. Здесь в строках 80—130 заготовлено всего 5 «новых» символов (см. табл. 13 и рис. 18).

Рис. 18

Таблица 13

«Новые» символы и их коды для примера 21

Коды	А	К	Л	С	5
«Новые» символы	А	К	Л	С	

Изображение, представленное на рисунке 18, получается при последовательной дешифровке значения строковой переменной TE$. Реализуется это действие строками 170—200. Одновременно формируется подпрограмма вывода P$. Если бы мы взглянули на значение P$ перед выполнением DRAW, то увидели бы следующую последовательность команд GML:

XA$(235);BR=L;XA$(236);BR=L;XA$(225);BR=L;
XA$(243);BR=L;XA$(243);BR=L;XA$(53);BR=L; (4)

предписывающих выполнить такие действия.

а) XA$ (t); (t∈{235, 236, 225, 243, 53}) — вывести «новый» символ, код ASCII которого равен t;

б) BR=L; — перевести курсор без рисования на L единиц вправо (организация пробелов между символами).

Несколько слов о том, как получено (4). В разъяснении, по-видимому, нуждается лишь 180-я программная строка. Ее выполнение можно представить в 4 этапа:

1. α=MID$(TE$, k, 1) — получение k-го символа TE$ (код одного из «новых» знаков),

2. β=ASC (α) — нахождение числового значения кода ASCII для α;

3. γ = STR$ (β) — представление кода ASCII для α в символьном виде;

4. Z$=MID$(γ, 2) — удаление 1-й знаковой позиции у γ.

2. 9. Распечатка экранного кадра

В реальных ситуациях довольно часто возникает необходимость в получении твердой копии изображения, находящегося в данный момент на экране. Для организации непосредственной распечатки

экранного кадра через принтер на лист бумаги средств Бейсика недостаточно. Однако нетрудно написать компактную программу на ассемблере Z-80, которая будет выполнять эту процедуру. Поскольку обучение ассемблеру не является целью данного пособия, мы ограничимся тем, что приведем программу на Бейсике, которая содержит в командах DATA соответствующие машинные коды для распечатки кадров и при запуске загружает эти коды в определенный участок памяти.

П Р И М Е Р 22. Распечатка экранного кадра

```
56000 'РАСПЕЧАТКА ЭКРАННОГО КАДРА (экран 2)
56001 '-------------------------------
56002 DEFUSR=&HEF00: RESTORE 56007
56003 FOR R=0 TO 171: READ R$
56004 POKE &HEF00+R,VAL("&H"+R$)
56005 NEXT R:R=USR(0):RETURN
56006 '-------подпрограмма в машинных кодах
56007 DATA 21,00,20,22,AC,EF,21,91,EF,CD,87
56008 DATA EF,21,9B,EF,CD,87,EF,0E,08,2A,AC
56009 DATA EF,CD,4A,00,57,E6,0F,5F,CB,3A,CB
56010 DATA 3A,CB,3A,CB,3A,00,3E,0F,BA,20,0D
56011 DATA BB,20,03,AF,18,11,CD,7F,EF,EE,FF
56012 DATA 18,0A,BB,28,04,3E,FF,18,03,CD,7F
56013 DATA EF,23,22,AC,EF,06,08,21,AE,EF,CB
56014 DATA 27,CB,1E,23,10,F9,0D,20,BE,06,08
56015 DATA 21,AE,EF,7E,CD,A5,00,23,10,F9,2A
56016 DATA AC,EF,AF,BD,20,A9,3E,0D,CD,A5,00
56017 DATA 3E,0A,CD,A5,00,3E,3B,BC,20,94,21
56018 DATA A2,EF,CD,87,EF,C9,CB,AC,CD,4A,00
56019 DATA CB,EC,C9,7E,FE,FF,C8,CD,A5,00,23
56020 DATA 1B,F6,1B,54,31,36,1B,45,1B,3E,0E
56021 DATA FF,1B,53,30,32,35,36,FF,1B,4E,0F
56022 DATA 1B,41,1B,3C,0D,0A,FF
56023 '-------- имя программы на диске "КАДР"
```

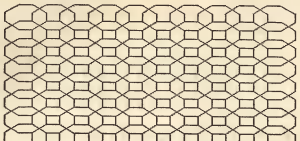


Рис. 19

Пусть программа примера 22 зафиксирована на диске в кодах ASCII под именем КАДР. Делается это командой SAVE "КАДР", А, где последняя буква А — латинская. Как воспользоваться программой КАДР? Если в основной программе α нет строк с номерами, бо́льшими 55999, и требуется распечатывать кадры, то осуществить это можно так.

а) Загружаем α в оперативную память, вставляем в требуемые точки обращение к подпрограмме КАДР: GOSUB 56000

б) Производим слияние программ α и КАДР

MERGE "КАДР"

в) Запускаем модифицированную программу α на счет и получаем требуемые распечатки.

Заметим, что спрайты (см. раздел 5) указанным способом с экрана на распечатку не копируются.

Рассмотрим пример. Пусть требуется вывести на экран 2 рисунок по фрагменту программы α примера 18 и получить его распечатку. Для этого осуществим такие действия. Загрузим α в память, добавим к ней строку обращения 55 (55 GOSUB 56000) и подпрограмму КАДР (MERGE "КАДР".). После запуска программы на счет получим распечатку орнамента, представленного на рисунке 19.

3. УПРАВЛЕНИЕ ВЫВОДОМ НА ПРИНТЕР

Изложение ориентировано на принтер GEMINI-10XR, относящийся к ударным точечно-матричным устройствам, широко применяемым в составе мини- и микроЭВМ. Нужное очертание символа воспроизводится ударами штырьков печатающей головки через красящую ленту по бумаге. Скорость печати—до 120 символов в секунду.

Принтер предназначен для вывода текстовой и графической информации. Предоставляемый им набор услуг весьма обширен. Возможна печать текста с изменением шага и наклона символов, выделением фрагментов полужирным шрифтом, расширением символов. Допускаются подчеркивания в тексте, а также использование верхних и нижних индексов. Организовано управление высотой строки и форматом выводимого текста. Разрешается конструирование собственных знаков. Имеются средства для вывода графической информации. Думается, что в ближайшем будущем такой набор услуг станет минимально возможным для любого принтера.

3.1. CHR$-коды

В дальнейшем мы часто будем встречаться с функцией

CHR$ (n) (n∈{0,1, ..., 255})

Значением CHR$ (n) является или некоторый управляющий сигнал, или символ, десятичный код ASCII которого равен n. Наиболее употребительный код CHR$ (27) называют кодом перехода. Введем для него и специальное обозначение Ф:

Ф = CHR$ (27)

Во многих случаях управление выводом реализуется последовательностью нескольких CHR$-кодов или CHR$-кодов и букв. Активизация управляющих сигналов организуется программно. Для этого их CHR$-коды делают операндами в команде LPRINT. Приведем некоторые примеры.

П Р И М Е Р 1. CHR$-КОДЫ В КОМАНДЕ LPRINT

```
10 LPRINT CHR$(27);"G"
20 LPRINT CHR$(7)
30 LPRINT CHR$(27);"R03";"*"
40 Z$=CHR$(27)
50 LPRINT Z$;"+";Z$;"P";CHR$(14);CHR$(0)
```

Отметим следующие 3 обстоятельства.

1. Не любые группы комбинаций CHR$-кодов допустимы.

2. Большинство CHR$-назначений, организованные командами LPRINT, справедливы и при выполнении команды LLIST.

3. CHR$-коды действуют именно так, как они описаны ниже, лишь при стандартном положении всех 12 двухпозиционных переключателей принтера.

3.2. Изменение шага печати

Шагом печати называют ширину прямоугольника, отводимого для вывода одиночного символа.

Допускается 6 различных шагов печати: 3 основных и 3 их расширения в два раза. Основные шаги имеют такие названия: «Цицеро», «Элите» и «Уплотненный». При включении принтер автоматически настраивается на печать «Цицеро» по 10 символов на дюйм, что позволяет на странице стандартных размеров (210 мм в ширину) размещать в строке до 80 знаков. При печати «Элите» на дюйм приходится 12 знаков и в строке размещается до 96 символов. При «уплотненной» печати на дюйм приходится 17 знаков и в строке размещается до 136 символов. Любой из трех основных шагов при необходимости можно удваивать. Получается так называемая расширенная печать, при которой выводимые символы в два раза шире обычных.

Задание печати с требуемым основным шагом, его расширение в два раза или отмена этого расширения производятся управляющими кодами, указанными в таблице 14.

Таблица 14

Управляющий код	Название шага и (или) выполняемые действия
1 Ф; "P" или Ф; "N"	«Цицеро». 10 знаков на дюйм. Шаг задан по умолчанию
2 Ф; "E"	«Элите». 12 знаков на дюйм
3 Ф; "Q"	«Уплотненный» шаг. 17 знаков на дюйм
4 CHR $ (14)	Удвоение действующего основного шага печати
5 CHR $ (15)	Отмена удвоенного шага печати

ПРИМЕР 2. ВЫБОР ШАГА ПЕЧАТИ

```
10 LPRINT CHR$(27);"P";"СТУДЕНТ"   run
20 LPRINT CHR$(27);"E";"СТУДЕНТ"   СТУДЕНТ
30 LPRINT CHR$(27);"Q";"СТУДЕНТ"   СТУДЕНТ
40 LPRINT CHR$(14);                СТУДЕНТ
50 LPRINT CHR$(27);"P";"СТУДЕНТ"   С Т У Д Е Н Т
60 LPRINT CHR$(27);"E";"СТУДЕНТ"   С Т У Д Е Н Т
70 LPRINT CHR$(27);"Q";"СТУДЕНТ"   СТУДЕНТ
80 LPRINT CHR$(27);"P";CHR$(15)
```


Рядом приведена полученная по программе распечатка. Обратите внимание на 80-ю программную строку. По ней восстанавливается основной тип печати «Цицеро» и отменяется расширение. Можно было бы и по-иному «сбросить» все предыдущие CHR$-назначения, применив код

Ф; "c1" (с — малое, латинское).

При этом принтер переведется в такое же состояние, которое он получает при включении питания. Более подробно об этом см. в пункте 3.8.

3.3. Выделение фрагментов

Для того чтобы выделить в текстах отдельные слова или фрагменты, применяются различные приемы. Основные из них такие: использование наклонного, полужирного или жирного шрифтов и подчеркивания. Наклонная печать называется «Италик». При ней символы распечатываются в слегка наклоненном вправо состоянии. Полужирная печать получается при двойном выводе каждого символа с небольшим смещением от исходного положения или вниз, или вправо. Жирная печать организуется при выводе каждого символа со смещением от исходного положения и вниз, и вправо. Подчеркивания проводятся снизу текста. Управляющие коды для выделения фрагментов при печати приведены в таблице 15.

Таблица 15

Управляющий код	Название и (или) выполняемые действия
1 Ф; "4"	Шрифт «Италик». Включение режима наклонной печати
2 Ф; "5"	Отмена режима наклонной печати
3 Ф; "G"	Полужирный шрифт. Включение режима печати двойным ударом со смещением изображения вниз
4 Ф; "H"	Отмена режима Ф; "G"
5 Ф; "I"	Полужирный шрифт. Включение режима печати двойным ударом со смещением изображения вправо
6 Ф; CHR $ (34)	Отмена режима Ф; "I"
7 Ф; "X"	Включение режима печати с подчеркиванием
8 Ф, "Y"	Отмена режима Ф; "X"

Двойной управляющий код

Ф;"!";Ф;"G"

определяет режим жирного шрифта. Его отмена реализуется комбинацией кодов

Ф; CHR$(34);Ф;"H"

Заметим, что режимы выделения можно использовать при любом шаге печати и комбинировать их с другими средствами вывода текстов.

П Р И М Е Р 3. ВЫДЕЛЕНИЕ ФРАГМЕНТОВ ПРИ ПЕЧАТИ

```
10 LPRINT CHR$(27);"4";"ДИСПЛЕЙ"
20 LPRINT CHR$(27);"5";
30 LPRINT CHR$(27);"!";"ДИСПЛЕЙ"
40 LPRINT CHR$(27);CHR$(34);
50 LPRINT CHR$(27);"X";"ДИСПЛЕЙ"
60 LPRINT CHR$(14);
70 LPRINT CHR$(27);"P";"ДИСПЛЕЙ"
80 LPRINT CHR$(27);"c1"
run
```

ДИСПЛЕЙ
ДИСПЛЕЙ
ДИСПЛЕЙ
ДИСПЛЕЙ

В распечатке имеем 4 строки. Из текста программы и таблиц 14 и 15 нетрудно увидеть, при каком типе выделения и каком шаге они получены.

3.4. Индексирование

Символы, располагаемые в верхней или нижней части строки в половину своего размера по высоте, называются соответственно верхними и нижними индексами. Печать индексов, или, по-другому, индексирование, организуется специальными управляющими кодами (см. таблицу 16). Наличие кода, позволяющего возвращать печатающую головку на 1 позицию назад, обеспечивает возможность вывода на одно знакоместо сразу и верхнего, и нижнего индексов. Установленный тип индексирования действует до нового назначения или полной отмены.

Таблица 16

Управляющий код	Выполняемые действия
1 Ф, "s1"	Включение режима верхнего индексирования
2 Ф; "s2"	Включение режима нижнего индексирования
3 Ф; "s0"	Отмена режимов индексирования
4 CHR $ (8)	Возврат головки принтера на 1 позицию назад

П Р И М Е Р 4. ИНДЕКСИРОВАНИЕ

```
10 U$=CHR$(27):T$="ТЕКСТ"
20 W$="ВЕРХНИЙ ИНДЕКС":N$="НИЖНИЙ ИНДЕКС"
30 LPRINT U$;"P";T$;U$;"s1";W$
40 LPRINT U$;"s0";T$;U$;"s2";N$
50 LPRINT U$;"s0";"S =";:A=7
60 FOR K=1 TO A .
70 LPRINT U$;"s0";" X";U$;"s2";CHR$(8);K;
80 IF K<>A THEN LPRINT U$;"s0";"+";
90 NEXT K:LPRINT "  "
100 LPRINT U$;"s0";"T =";
110 FOR K=1 TO A
120 LPRINT U$;"s0";" Y";U$;"s1";CHR$(8);K;
130 IF K<>A THEN LPRINT U$;"s0";"+";
140 NEXT K
150 LPRINT U$;"c1"
run
```

ТЕКСТ$_{ВЕРХНИЙ\ ИНДЕКС}$
ТЕКСТ$_{НИЖНИЙ\ ИНДЕКС}$

$$S = X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7$$
$$T = Y^1 + Y^2 + Y^3 + Y^4 + Y^5 + Y^6 + Y^7$$

Распечатка для примера 4 приведена за текстом программы.

3.5. Изменение высоты строк

При включении принтера высота строки автоматически устанавливается из расчета 6 строк на дюйм. Изменяется она в любой момент времени с помощью кодов, приведенных в таблице 17.

Таблица 17

Управляющие коды	Выполняемые действия
1 Ф; "A"	Установка высоты строки в 1/6 дюйма
2 Ф; "B"	Установка высоты строки в 1/9 дюйма
3 Ф; "T"; "n"	Установка высоты строки в n/144 дюйма. Здесь n — целое число, которое необходимо задавать двумя цифрами: 01, 02, ..., 99.

Варьируя параметр n, можно подобрать требуемую высоту строки. В следующем примере при основном шаге «Цицеро» распечатка получена при высотах 1/6, 1/4 и 1/3 дюйма.

П Р И М Е Р 5. ИЗМЕНЕНИЕ ВЫСОТЫ СТРОКИ

```
10 LPRINT CHR$(27);"A";:GOSUB 60
20 LPRINT CHR$(27);"T";"36";:GOSUB 60
30 LPRINT CHR$(27);"T";"48";:GOSUB 60
40 LPRINT CHR$(27);"c1":END
60 LPRINT "ПРИ ВЫБОРЕ ВЫСОТЫ СТРОКИ"
70 LPRINT "ПАРАМЕТР   n   ЗАДАВАЙТЕ"
80 LPRINT "В ВИДЕ: 01, 02, .... 99!"
85 LPRINT "-------------------------"
90 RETURN
```

```
ПРИ ВЫБОРЕ ВЫСОТЫ СТРОКИ
ПАРАМЕТР   n   ЗАДАВАЙТЕ
В ВИДЕ: 01, 02, ..., 99!
-------------------------
ПРИ ВЫБОРЕ ВЫСОТЫ СТРОКИ

ПАРАМЕТР   n   ЗАДАВАЙТЕ

В ВИДЕ: 01, 02, ..., 99!

-------------------------
```

3.6. Установка полей на странице

Как и в любой пишущей машинке, в принтере можно регулировать ширину текста и его расположение на странице. По управляющим кодам (см. табл. 18) независимо друг от друга устанавливается ширина левого и правого полей. В дальнейшем вывод будет осу-

Таблица 18

Управляющие коды	Выполняемые действия
1 Ф; "L"; "n"	Установка левого поля с позиции n (n ∈ {000, 001, ...})
2 Ф; "/"; "n"	Установка правого поля с позиции n (n ∈ {000, 001, ...})

ществляться между этими полями. Отмена зафиксированной разметки листа производится или новым ее назначением, или общей инициализацией принтера кодом Ф;"c1".

В распечатке приведенного ниже примера один и тот же текст выведен двумя различными способами.

П Р И М Е Р 6. УСТАНОВКА ПОЛЕЙ

```
10 T$="ЗАДАВАЙТЕ n В ВИДЕ:"
20 K$="000, 001, 002, ...!":GOSUB 70
30 LPRINT CHR$(27);"L";"010";:GOSUB 70
40 LPRINT CHR$(27);"L";"020";
50 LPRINT CHR$(27);"/";"030";:GOSUB 70
60 LPRINT CHR$(27);"c1":END
70 LPRINT T$:LPRINT K$;CHR$(10):RETURN
run
```

```
ЗАДАВАЙТЕ n В ВИДЕ:
000, 001, 002, ...!

          ЗАДАВАЙТЕ n В ВИДЕ:
          000, 001, 002, ...!
```

3.7. Горизонтальная табуляция

На пишущих машинках для расположения данных при печати с конкретных позиций предусмотрен механизм табуляции. Аналогичные средства имеются и для принтера. Для установки горизонтальных меток табуляции используется код

Ф; "("; "n_1, n_2, ...n_k." (1)

где n_s ($s=\overline{1,k}$) — номера позиций в строке (см. табл. 19). Величины n_s называются *метками табуляции*. Задаваться они должны целыми трехзначными числами с обязательным указанием количества сотен, десятков и единиц и располагаться в коде (1) в порядке возрастания. Обратите внимание на разделители в коде (1). Между элементами n_s ставятся запятые, а после n_k обязательно фиксируется точка. Например, Ф; "("; "005, 020, 050, 070."

Если метки не устанавливались или стерты, то по умолчанию $n_1=0$, $n_2=10$, $n_3=20$, ...

Назначение горизонтальной табуляции само по себе ни к каким действиям не приводит. Для использования меток n_s требуется задать код

CHR$ (9) (2)

осуществляющий разовое перемещение печатающей головки принтера на следующую метку. Специфика рассматриваемого принтера требует предварить использование кодов (2) выполнением команды

POKE&HF418,1

Таблица 19

Горизонтальные метки табуляции

Управляющие коды	Выполняемые действия
1 Ф; "("; n_1, n_2, ... n_k"	Установка меток табуляции n_1, n_2, .., n_k
2 Ф;")";"m_1, m_2, ... m_p"	Выборочное стирание меток табуляции из позиций: m_1, m_2, ..., m_p
3 Ф; "2"	Стирание всех меток табуляции, установленных пользователем
4 CHR $ (9)	Перемещение головки принтера на следующую правую метку
5 Ф; "fl"; CHR $ (n)	Перемещение головки принтера на n позиций по строке вправо

П Р И М Е Р 7. УСТАНОВКА ГОРИЗОНТАЛЬНЫХ МЕТОК

```
10 LPRINT CHR$(27);"(";"005,013,025."
15 T=3:' T - количество меток
20 POKE &HF418,1
30 FOR K=1 TO 30:READ X
40 LPRINT CHR$(9);X;
50 IF KMODT=0 THEN LPRINT
60 NEXT K
70 LPRINT CHR$(27);"c1"
80 DATA 345,12,2454,34,4,5,689,7,8,987
90 DATA -250,-1457,2,3,4,5699,6,7,8,9
100 DATA 0,13,247,3215,4,5,6,734,86,95
run
      345        12          2454
      34         4           5
      689        7           8
      987       -250        -1457
```



```
      2          3           4
      5699       6           7
      8          9           0
      13         247         3215
      4          5           6
      734        86          95
```

Вслед за программой приведена полученная по ней распечатка. Данные из блока, сформированного командами 80—100, выведены в 3 колонки в соответствии с метками табуляции: 005, 013 и 025. В чем смысл команды 50? Дело в том, что когда головка принтера попадает за последнюю метку табуляции n_k, то CHR$ (9) перестает действовать. Поэтому после вывода очередных 3 чисел необходимо искусственно возвратить головку в начало следующей строки. Это и делает команда 50.

Из таблицы 19 мы видим, что имеются коды для выборочного или полного удаления горизонтальных меток табуляции, установленных пользователем. Их синтаксис и выполняемое действие после детального обсуждения кода (1), по-видимому, ясны. Отметим лишь, что параметры m_s ($s = \overline{1,p}$) должны задаваться целыми трехзначными числами так же, как и n_s в коде (1).

Далее, код, определяющий перемещение головки принтера на n позиций по строке, равнозначен функции SPACE$ (n), формирующей n пробелов.

3.8. Дополнительные управляющие коды

Кроме ранее рассмотренных, имеется еще целый ряд дополнительных кодов, облегчающих работу с принтером. В приложении (см. табл. 1) они указаны под номерами 28—51. Некоторые из них использованы в примере 8.

П Р И М Е Р 8. НЕКОТОРЫЕ ДОПОЛНИТЕЛЬНЫЕ КОДЫ

```
10 '-----------------------------------звонок
20 Z$=CHR$(27):LPRINT CHR$(7);
30 '---------------------повтор символа 30 раз
40 LPRINT CHR$(27);"R030";"H";
50 '-------------------------протяжка на 2 строки
```

```
60 LPRINT CHR$(27);"a";CHR$(2);
70 '----------------------повтор символа 20 раз
80 LPRINT CHR$(27);"R020";"Z
90 '--------------------пропуск точечных позиций
100 GOSUB 160:LPRINT Z$;"F";"0001";:GOSUB 160
110 LPRINT Z$;"F";"0002";:GOSUB 160
120 LPRINT Z$;"F";"0003";:GOSUB 160
130 LPRINT Z$;"F";"0050";:GOSUB 160
140 GOSUB 160:LPRINT Z$;"F";"0100";:GOSUB 160
150 END
160 LPRINT "ПРОПУСК ТОЧЕЧНЫХ ПОЗИЦИЙ": RETURN
```

3.9. Макрокоды

Для того чтобы программы управления печатью были более компактными и наглядными, можно использовать при их оформлении специальный механизм макроопределений. О нем и пойдет речь ниже.

Последовательную группу управляющих кодов, записанную в виде:

Ф; "+"; коды; CHR$ (0)

называют макроопределением (макрокодом или просто МАКРО).

Одно макроопределение может содержать до 16 кодов. Оно не воздействует на печать в процессе выполнения программы, пока не будет активизировано макрокомандой Ф; "%" (см. табл. 20).

Таблица 20

Управляющие коды	Название и выполняемые действия
1 Ф; "+"; коды; CHR $ (0)	Макроопределение, представляющее собой совокупность управляющих кодов
2 Ф; "%"	Макрокоманда, активизирующая последнее из выполненных макроопределений

В программе могут встретиться несколько макроопределений. Действует всегда последнее выполненное МАКРО.

Для уяснения принципа действия МАКРО предлагается проанализировать следующий пример, где распечатка приводится сразу вслед за программой.

ПРИМЕР 9. МАКРООПРЕДЕЛЕНИЕ

```
 5'----------------------МАКРООПРЕДЕЛЕНИЕ
10 LPRINT CHR$(27);"+";
20 LPRINT CHR$(27);"P";CHR$(14);
30 LPRINT CHR$(27);"(";"003,012,020."
40 LPRINT CHR$(0)
50  '-------------------------------------
60 LPRINT CHR$(27);"%":GOSUB 110
70 LPRINT CHR$(15);CHR$(27);"X":GOSUB 110
80 LPRINT CHR$(27);"%":GOSUB 110
90 LPRINT CHR$(27);"c1":END
100 '-------------------------подпрограмма
110 POKE &HF418,1:T=2:RESTORE 190
120 LPRINT:LPRINT "ЯЗЫКИ"
180  '-------------------------------------
130 LPRINT "ПРОГРАММИРОВАНИЯ:"
140 FOR K=1 TO 8
150 READ X$:LPRINT CHR$(9);X$;
160 IF KMODT=0 THEN LPRINT
170 NEXT K:RETURN
190 DATA БЕЙСИК,АЛГОЛ,СИ,АДА
200 DATA ЛИСП,ПАСКАЛЬ,ПЛ/1,ФОРТРАН
run


ЯЗЫКИ
ПРОГРАММИРОВАНИЯ:
        БЕЙСИК          АЛГОЛ
        СИ              АДА
        ЛИСП            ПАСКАЛЬ
        ПЛ/1            ФОРТРАН
```

ЯЗЫКИ ПРОГРАММИРОВАНИЯ:

БЕЙСИК	АЛГОЛ
СИ	АДА
ЛИСП	ПАСКАЛЬ
ПЛ/1	ФОРТРАН

ЯЗЫКИ ПРОГРАММИРОВАНИЯ:

БЕЙСИК	АЛГОЛ
СИ	АДА
ЛИСП	ПАСКАЛЬ
ПЛ/1	ФОРТРАН

Механизм МАКРО является специфической формой организации подпрограмм. Однако стоит заметить, что использование стандартного механизма подпрограмм во многих случаях более естественно.

3.10. Создание новых символов

Принтер GEMINI-10XR дает возможность пользователю создавать новые символы, загружать их коды в специально отведенное для этого место памяти (RAM), а затем выводить их на печать. Единовременно в RAM может храниться до 94 различных знаков, а общее их количество равно 2×89^7. Прежде чем приступить к разговору о формировании новых символов, нам предстоит более детально ознакомиться с работой данного принтера и принятым в нем способом кодирования знаков.

Любой символ алфавита формируется принтером на листе бумаги с помощью 9 тонких штырьков (иголок) печатающей головки. Штырьки расположены в обойме в одном вертикальном ряду, и каждый из них управляется своим магнитом. В процессе печати на эти магниты подаются потенциалы и в зависимости от их величины соответствующие штырьки либо втягиваются внутрь обоймы, либо выталкиваются из нее и ударяют по красящей ленте. Каждый такой удар оставляет на листе бумаги небольших размеров круглое пятнышко цвета красителя ленты. Будем условно называть это пятнышко точкой. Тогда можно сказать, что любой символ алфавита

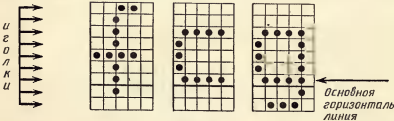


Рис. 20

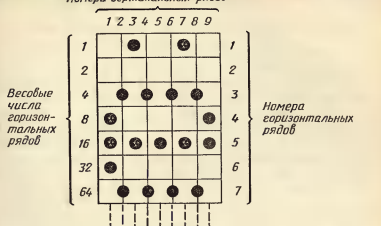


Рис. 21

рисуется отдельными точками. Например, из рисунка 20 (схематическое изображение некоторых символов в строке) видно, что буква С состоит из 11 точек.

Из сказанного ясно, почему принтер GEMINI-10XR относят к точечно-матричным печатающим устройствам ударного типа.

В одной строке выводимых символов условно можно выделить 9 горизонтальных рядов. Конкретная иголка участвует в создании точек своего ряда. Но любой символ, в том числе и заглавные буквы, должен полностью располагаться в 7 верхних или 7 нижних рядах. При формировании конкретного символа для точек доступны 9 вертикальных рядов. Не допускается лишь, чтобы по горизонтали оказались рядом две соседние точки. Таким образом, если взять сетку размера 7×5 и отождествить ее строки с горизонтальными рядами, а столбцы и разделяющие их линии — с вертикальными рядами, то легко нарисовать на ней любой символ алфавита Бейсика (см. рис. 21).

По типу расположения в выводимой строке символы можно разделить на две группы: те, которые не опускаются ниже основной горизонтальной линии, и значит, в их создании не принимают участия две нижние иголки (см. рис. 20), и все остальные. Первые будем называть символами без понижения и приписывать им характеристику $m_0 = 0$, а вторые — символами с понижением и

приписывать им характеристику $m_0 = 1$. Например, для буквы «ё» $m_0 = 0$.

Параметр m_0 одна из компонент специального кода

$$m_0,\ m_1,\ m_2,\ ...,\ m_9 \qquad (3)$$

любого, в том числе и нового, знака γ. Как сформировать (3)? Пусть γ зафиксирован на сетке. Припишем ее горизонтальным рядам в порядке их следования весовые числа: 1, 2, 4, ..., 64 (см. рис. 21). Подсчитываем значения величин $m_s (s = \overline{1, 9})$, равные сумме весовых чисел тех горизонтальных рядов, в которых для вертикали s проставлены точки. Код (3) получен. Описанная процедура весьма наглядно просматривается на рисунке 21 для символа «ё».

Коды (3) всех стандартных символов хранятся в запоминающем устройстве принтера ROM. Кроме того, как мы уже отмечали, коды вновь создаваемых знаков загружаются в область памяти, именуемую RAM. Итак, пусть для знака γ получена последовательность (3). По управляющему коду (см. табл. 21):

Ф; ″,″; CHR$ (n); CHR$ (m_0); CHR$ (m_1)... CHR$ (m_9)

γ загружается в ROM, где: n — десятичный код, приписываемый γ (n∈{33, 34, ..., 126}); m_0 — характеристика понижения γ; $m_s (s = \overline{1, 9})$ — весовые числа вертикальных рядов γ.

Далее, чтобы напечатать γ = CHR$ (n), необходимо включить режим вывода новых символов. Делается это по коду

Ф; ″,″ (″,″ = CHR$ (39))

Отмена режима Ф; ″″ производится по любому из кодов

Ф; ″&″ или Ф; ″$″. (4)

При выполнении (4) CHR$ (n) опять становится стандартным символом алфавита с кодом ASCIIn. Таким образом мы одновременно можем печатать и новый знак γ, и соответствующий ему стандартный символ.

Таблица 21

Управляющие коды	Выполняемые действия
1 Ф; ″,″; CHR $ (n); CHR $ (m_0); CHR $ (m_1); ... ... CHR $ (m_9)	Загрузка нового символа γ в память принтера RAM. Здесь: 1. n — код γ (n ∈ {33, 34, ..., 126}) 2. m_0 — характеристика понижения γ ($m_0 \in \{0, 1\}$) 3. m_s ($s = \overline{1, 9}$) — весовые числа вертикальных рядов γ
2 Ф; ″,″	Включение режима вывода новых символов
3 Ф: ″&″ или Ф; ″$″	Отмена режима Ф; ″″

ПРИМЕР 10. НОВЫЙ СИМВОЛ ∀

```
10 Z$=CHR$(27):LPRINT Z$;"P";CHR$(14)
20 ' -------- формирование нового знака
30 DATA 0,15,16,36,64,4,64,36,16,15
40 LPRINT Z$;",";CHR$(98);
50 FOR K=1 TO 10
60 READ M:LPRINT CHR$(M);
70 NEXT K
80 ' ---------------------- распечатки
90 LPRINT Z$;"'";:GOSUB 150
100 LPRINT Z$;"&";:GOSUB 150
110 LPRINT Z$;"'";:GOSUB 170
120 LPRINT Z$;"$";:GOSUB 170
130 ' ------------------------------------
140 LPRINT Z$;"c1";:END
150 FOR K=1 TO 15:LPRINT CHR$(98);:NEXT
160 LPRINT:RETURN
170 FOR K=1 TO 15:LPRINT "b";:NEXT
180 LPRINT:RETURN
run
```

```
∀∀∀∀∀∀∀∀∀∀∀∀∀∀∀
bbbbbbbbbbbbbbb
∀∀∀∀∀∀∀∀∀∀∀∀∀∀∀
bbbbbbbbbbbbbbb
```

В этом примере сформирован квантор ∀. После включения режима печати новых символов он выводится и по коду 98, и с помощью символа ″b″ (строки 70 и 90). Отмена режима Ф; ″″ приводит к выводу стандартного символа ″b″ (строки 80 и 100).

Заметим, что новые знаки исчезают из памяти RAM лишь в случаях, если на их место загружаются новые символы или выключается принтер.

```
ПРИМЕР 11. ДВА НОВЫХ СИМВОЛА ∀ и ∃
10 Z$=CHR$(27):J$=CHR$(95):I$=CHR$(96)
20 D$="x,y- вещественные":T$="положительные"
30 LPRINT Z$;"P";CHR$(14);
40 '-------------формирование новых знаков
50 LPRINT Z$;",";J$;:GOSUB 140
60 LPRINT Z$;",";I$;:GOSUB 140
70 '-------------------------распечатки
80 LPRINT Z$;"'";D$;",":LPRINT "      ";T$
90 LPRINT:LPRINT "n   - натуральное":LPRINT
100 LPRINT"АКСИОМА АРХИМЕДА: "
110 LPRINT "  ";J$;"x";J$;"y";I$;"n(x≤n·y)"
120 '-----------------------------------
130 LPRINT Z$;"c1";:END
140 FOR K=1 TO 10
150 READ M:LPRINT CHR$(M);
160 NEXT K:RETURN
170 DATA 0,15,16,36,64,4,64,36,16,15
180 DATA 0,65,0,73,0,73,0,73,0,127
run
x,y- вещественные,
     положительные
n    - натуральное
АКСИОМА АРХИМЕДА:
   ∀x∀y∃n(x≤n·y)
```

3.11. Растровая графика

По аналогии с созданием и выводом новых символов рассматриваемый принтер может осуществлять печать произвольных изображений: картин, схем, карт, графиков и т. п. Подобная растровая (поточечная) графика допускает разрешающую способность до 160 точек на дюйм. Это существенно лучше обычного текстового режима, при котором все, в том числе и созданные пользователем, символы выводятся при разрешающей способности в 60 точек на дюйм.

Для изображения λ, которое подлежит в будущем распечатке, необходимо создать специальный цифровой код. Опишем, как это делается.

Пусть λ представлено на прямоугольной сетке с m горизонтальными и n вертикальными рядами, где m кратно 8. Каждая строка, состоящая из 8 последовательных горизонтальных рядов, будет кодироваться отдельно, выводиться без всяких понижений и формироваться первыми 8 штырьками головки принтера. Зафиксируем строку i и припишем ее последовательным горизонтальным рядам весовые числа: 1, 2, 4, ..., 128. По ним так же, как это делалось в предыдущем пункте, находим весовые числа вертикальных рядов

$$m_{i1},\ m_{i2},\ \ldots,\ m_{in}\ (i=1,\ 2,\ \ldots) \qquad (5)$$

Каждое $m_{is}(s=\overline{1,\ n})$ равно сумме весовых чисел тех горизонтальных рядов, в которых для вертикали s проставлены точки изображения λ. Стилизованное изображение символа Р на сетке и его некоторые характеристики см. на рисунке 22.

Последовательность (5) является кодом t_i строки i. Построив все $t_i\,(i=\overline{1,\ m/8})$, получим матрицу

$$(m_{ij})\ (i=\overline{1,\ m/8};\ j=\overline{1,\ n}).$$

Вывод изображения λ реализуется построчно по управляющим кодам

$$\Phi;\ "S";\ "n";\ \mathrm{CHR\$}\ (m_{i1});\ \mathrm{CHR\$}\ (m_{i2});\ \ldots\ \mathrm{CHR\$}\ (m_{in}) \qquad (6)$$
$$(i=\overline{1,\ m/8}).$$

где n — количество вертикальных рядов λ ($n\in\{0001,\ 0002,\ \ldots\}$),

m_{is} — весовое число вертикального ряда $s\,(s=\overline{1,\ n})$ для строки

$$i\,(i=\overline{1,\ m/8}).$$

При использовании управляющего кода (6) для слитного восприятия λ высоту строки требуется установить в 16/144 дюйма. Это делается кодом

$$\Phi;\ "T";\ "16"$$

Кроме того, специфика данного принтера требует предварить использование кода (6) выполнением команды

```
POKE&HF418,1
```

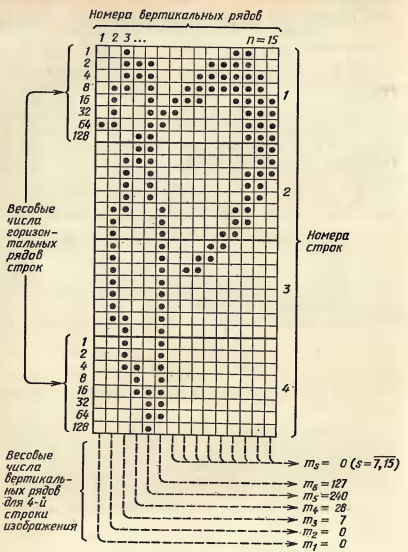


Рис. 22

ПРИМЕР 12. РАСТРОВАЯ ГРАФИКА. ВЫВОД СТИЛИЗОВАННОЙ БУКВЫ "Р"

```
10 Z$=CHR$(27):LPRINT Z$;"T";"16";CHR$(14)
20 POKE &HF418,1
30 FOR I=1 TO 4:LPRINT Z$;"S";"0015";
40 FOR J=1 TO 15:READ M:LPRINT CHR$(M);:NEXT
50 LPRINT:NEXT I
60 DATA 64,120,15,6,254:'-------------1 строка
70 DATA 96,48,24,28,14,14,31,255,252,240
80 DATA 0,224,62,3,31:'---------------2 строка
90 DATA 224,0,0,0,0,128,224,124,31,7
100 DATA 0,127,192,0,0:'--------------3 строка
110 DATA 255,0,4,6,3,1,0,0,0,0
120 DATA 0,0,7,28,240:'----------------4 строка
130 DATA 127,0,0,0,0,0,0,0,0,0
run
```


Таблица 22

Управляющие коды	Выполняемые действия
1. Ф; "s"; "n"; CHR $ (m_{i1}); CHR $ (m_{i2}); .... CHR $ (m_{in})	Вывод i-й строки изображения λ Здесь: 1. n — количество вертикальных рядов λ ($n \in \{0001, 0002, \ldots\}$) 2. m_{ik} — весовое число вертикального ряда $s(s=\overline{1, n})$ для строки i
2. Ф; "N"	Включение режима нормальной плотности печати — 80 точек на дюйм. Ф; "N" задан по умолчанию
3. Ф; "E"	Включение режима средней плотности печати — 90 точек на дюйм
4. Ф; "P" или Ф; "Q"	Включение режима двойной плотности печати — 160 точек на дюйм

Здесь символ «Р» печатается при шаге «Цицеро» с расширением в два раза за 4 прохода головки принтера (см. рис. 22). Подчеркнем, что если бы мы выводили не один знак, а более емкое изображение, то это не отразилось бы на ядре программы (строки 10—50). Разрослась бы только ее часть, связанная с данными.

3.12. Графики функций

Описанный в предыдущем пункте подход к использованию растровой графики не совсем удобен из-за большого количества

предварительных вычислений, которые фактически приходится выполнять «вручную». По-видимому, там, где это возможно, предпочтительнее формировать коды строк изображения с помощью компьютера. Например, это довольно просто делать при выводе графиков различных функций.

По программе «Графики» можно напечатать график произвольной явной функции действительного переменного X: Y=F (X). Для этого необходимо проделать такие операции:

а) В строках 710 и далее указать аналитическое выражение для F (X).

б) Запустить программу и ответить на запрос с экрана, в каком прямоугольнике R должен быть сформирован график F(X). Ввод четверки чисел (A, B, C, D) и определит

$$R=\left\{(X,\ Y)\ \middle|\ \begin{matrix}A\leqslant X\leqslant B\\ C\leqslant Y\leqslant D\end{matrix}\right\}$$

в) Ответить на запрос о размере распечатки. Это позволит зафиксировать в точках величину второго прямоугольника (кадра) T, в котором будет размещаться на бумаге изображение. Если (P, Q) — введенная пара чисел и P=16 * P; Q=16 * Q, то

$$T=\left\{(I,\ I)\ \middle|\ \begin{matrix}1\leqslant I\leqslant Q\\ 1\leqslant I\leqslant P\end{matrix}\right\}$$

г) Задать плотность распечатки U (1 или 2). При обоих значениях U количество точек в T одинаково, но при U=2 кадр T будет сжат в два раза вдоль оси абсцисс.

Описанная процедура позволяет получать графики довольно сложных функций. Непрерывность F(X) не предполагается. Некоторые примеры функций и их графиков, полученных по обсуждаемой программе, приведены на рисунке 23. Параметры ввода:

а) A=−8.5, Q=25,	B=8.1, P=11,	C=−5, U=1	D=7
в) A=−7, Q=25,	B=6, P=6,	C=−6, U=2	D=7
с) A=−7, Q=25,	B=7, P=6,	C=−2, U=2	D=2

```
ПРИМЕР 13. ПРОГРАММА "ГРАФИКИ"
10 ' ПЕЧАТЬ ГРАФИКА ФУНКЦИИ Y=F(X)
20 '-------------------------------------------------
30 COLOR 15,4:SCREEN 0:KEY OFF:WIDTH 40
40 DEFINT I,J,M,V,P,Q,L,R,E,Z
50 L$=STRING$(37,"-"):Z$=CHR$(27)
60 PRINT "          ПЕЧАТЬ ГРАФИКОВ ФУНКЦИЙ"
```

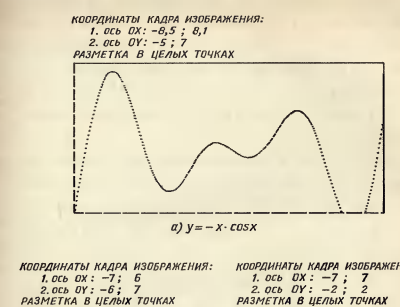



Рис. 23

```
70 PRINT L$
80 PRINT "КООРДИНАТЫ ПРЯМОУГОЛЬНИКА":PRINT
90 PRINT "1. интервал определения [A,B]"
100 INPUT "    A=";A:INPUT "    B=";B
110 PRINT
120 PRINT "2. интервал изменения   [C,D]"
130 INPUT "    C=";C:INPUT "    D=";D
140 PRINT
150 PRINT "РАЗМЕРЫ РАСПЕЧАТКИ":PRINT
160 PRINT "1. по горизонтали (Q=1,2,...,25)"
170 INPUT "    Q=";Q:Q=Q*16:PRINT
```

```
180 PRINT "2. по вертикали   (P=1,2,...,11)"
190 INPUT "     P=";P:L=P%2:P=P%16:PRINT
200 PRINT "ПЛОТНОСТЬ РАСПЕЧАТКИ (U=1,2)"
210 PRINT
220 INPUT "     U=";U:IF U<>1ANDU<>2 THEN 220
230 CLS
240 LOCATE 8,9:PRINT "проводятся вычисления"
250 '-----------------------------------ЯДРО
260 DIM M(L,Q),V(P):ON ERROR GOTO 690
270 K1=(B-A)/(Q-1):K2=(D-C)/(P-1):K5=1/K1
280 K6=1/K2:K3=A-K1:K4=C-K2:E=P\8+1
290 R$=RIGHT$("0000"+MID$(STR$(Q),2),4)
300 FOR I=0 TO L-1: FOR J=1 TO 8
310 V(I%8+J)=2^(8-J)
320 NEXT J,I
330 '-------------------------кадр изображения
340 FOR I=1 TO Q:M(1,I)=1:M(L,I)=128:NEXT
350 FOR J=1 TO L:M(J,1)=255:M(J,Q)=255:NEXT
360 '---------------------------разметка кадра
370 IF B-A<2 THEN 430
380 FOR X=A TO B+1:Z=X
390 IF Z<=A OR Z>=B THEN 420
400 I=(Z-K3)%K5
410 M(L,I-1)=0:M(L,I)=0:M(L,I+1)=0
420 NEXT X
430 IF D-C<2 THEN 550
440 FOR Y=C TO D+1:Z=Y
450 IF Z<=C OR Z>=D THEN 530
460 J=(Z-K4)%K6:R=E-(J+7)\8:M=V(J)
470 IF M<>1 THEN 490
```



```
480 M(R,1)=M(R,1)-3:M(R-1,1)=M(R-1,1)-128
490 IF M<>128 THEN 510
500 M(R,1)=M(R,1)-192:M(R+1,1)=M(R+1,1)-1
510 IF M=1 OR M=128 THEN 530
520 M(R,1)=M(R,1)-M-V(J-1)-V(J+1)
530 NEXT Y
540 '-----------------весовые числа для Y=F(X)
550 FOR I=2 TO Q-1:X=K1%I+K3:GOSUB 710
560 J=(Y-K4)%K6:IF J<2 OR J>=P THEN 580
570 R=E-(J+7)\8:M(R,I)=M(R,I)+V(J)
580 NEXT I:'-----------------ВЫВОД ИЗОБРАЖЕНИЯ
590 CLS
600 LPRINT "КООРДИНАТЫ КАДРА ИЗОБРАЖЕНИЯ:"
610 LPRINT "   1. ось OX: "A;" ";B
620 LPRINT "   2. ось OY: "C;" ";D
630 LPRINT "РАЗМЕТКА В ЦЕЛЫХ ТОЧКАХ"
640 LPRINT Z$;"N";:IF U=2 THEN LPRINTZ$;"P";
650 LPRINT Z$;"T";"16";Z$;"!";:POKE &HF418,1
660 FOR I=1 TO L:LPRINT Z$;"S";R$;
670 FOR J=1 TO Q:LPRINT CHR$(M(I,J));:NEXT J
680 LPRINT:NEXT:LPRINT Z$;"c1";CHR$(10):END
690 Y=D+10:ON ERROR GOTO 0:RESUME 740
700 '=========ПОДПРОГРАММА ДЛЯ ФУНКЦИИ Y=F(X)
710 'Y=F(X)
750 RETURN
```

Несколько слов о программе. Между точками $(X, Y) \in R$ и $(I, J) \in T$ по формулам

$$\begin{cases} X=(I-1)\cdot\frac{B-A}{Q-1}+A \\ Y=(J-1)\cdot\frac{D-C}{P-1}+C \end{cases} \qquad (7)$$

устанавливается взаимно однозначное соответствие. Но в T нас интересуют лишь целочисленные пары (I, J). Они и являются возможными координатами изображения. Чтобы при фиксированном параметре Q разместить в T максимально возможное количество точек F(X), в программе используется следующий алгоритм. Прежде всего для каждого I от 2 до Q—1 выполняются пункты 1—4.

1. По первой из формул (7) при фиксированном I находится аргумент X для F(X).

2. Вычисляется значение Y=F(X). Если (X, Y)$\notin$R или F не определена в точке X, то переходим к следующему I.

3. По второй из формул (7) при фиксированном Y находим J и округляем его до ближайшего целого. Если (I, J)$\notin$T, то переходим к следующему I.

4. Пара (I, J) есть точка изображения. Определяем для нее весовое число (см. пункт 3.11) и запоминаем его в массиве M.

Далее, полученный массив M фактически является матрицей изображения и выводится так, как это описано в пункте 3.11.

Кроме графика для F(X), программа рассчитывает и распечатывает рамку кадра, размеченную в целых точках. Делается это по операторам из строк 330—530. Если подобная рамка с разметкой в распечатке не требуется, то строки 330—530 из программы можно удалить. На существе дела это не отразится.

В заключение остановимся на таком моменте. Набор S встроенных стандартных функций Бейсика весьма ограничен. Поэтому, скажем, для функции

$$y=x+\arcsin(\sin x) \qquad (8)$$

непосредственно графика не построить. Но оказывается, многие элементарные функции могут быть выражены в виде комбинаций из S. В таблице 23 приведены соотношения между некоторыми элементарными функциями. Используя ее, мы смогли бы написать подпрограмму для вычисления (8) так:

```
710 Y=-SIN(X)^2+1      730 Y=X+ATN(Y)
720 Y=SIN(X)/SQR(Y)    740 RETURN
```

Таблица 23

Название функции	Выражение	Область определения
1 Секанс	$\sec x=1/\cos x$	$x\neq\pi/2+\pi k$, $k=0, \pm1, \pm2, \ldots$
2 Косеканс	$\csc x=1/\sin x$	$x\neq\pi k$, $k=0, \pm1, \pm2, \ldots$
3 Котангенс	$\operatorname{ctg} x=1/\operatorname{tg} x$	$x\neq\pi k$, $k=0, \pm1, \pm2, \ldots$
4 Арксинус	$\arcsin x=\operatorname{arctg}(x/\sqrt{1-x^2})$	$x\in[-1; 1]$

Продолжение

Название функции	Выражение	Область определения
5 Арккосинус	$\arccos x=\frac{\pi}{2}-\operatorname{arctg}(x/\sqrt{1-x^2})$	$x\in[-1; 1]$
6 Арккотангенс	$\operatorname{arcctg} x=\frac{\pi}{2}-\operatorname{arctg} x$	$x\in(-\infty; +\infty)$
7 Гиперболический синус	$\operatorname{sh} x=(e^x-e^{-x})/2$	$x\in(-\infty; +\infty)$
8 Гиперболический косинус	$\operatorname{ch} x=(e^x+e^{-x})/2$	$x\in(-\infty; +\infty)$
9 Гиперболический тангенс	$\operatorname{th} x=(e^x-e^{-x})/(e^x+e^{-x})$	$x\in(-\infty; +\infty)$
10 Гиперболический котангенс	$\operatorname{cth} x=(e^x+e^{-x})/(e^x-e^{-x})$	$x\neq0$
11 Гиперболический арксинус	$\operatorname{ash} x=\ln(x+\sqrt{x^2+1})$	$x\in(-\infty; +\infty)$
12 Гиперболический арккосинус	$\operatorname{ach} x=\ln(x+\sqrt{x^2-1})$	$x\geqslant1$
13 Гиперболический арктангенс	$\operatorname{ath} x=\frac{1}{2}\ln((1+x)/(1-x))$	$\lvert x\rvert<1$
14 Гиперболический арккотангенс	$\operatorname{acth} x=\frac{1}{2}\ln((1-x)/(1+x))$	$\lvert x\rvert>1$
15 Логарифм N по основанию a	$\log_a N=(\ln N)/(\ln a)$	$a, N>0$ $a\neq1$

Мы не касались вопроса о построении графиков функций, заданных параметрически:

$x=\varphi(t)$, $y=\psi(t)$ ($\alpha\leqslant t\leqslant\beta$), где φ, ψ — функции действительной переменной t; α, β — вещественные числа. Соответствующая программа может быть получена модификацией строк 550—580 фрагмента «Графика». При этом, разумеется, необходимо иметь подпрограмму вычисления значений $\varphi(t)$ и $\psi(t)$.

4. ФАЙЛЫ

4.1. Общие сведения

Термин *файл* происходит от английского слова *file*, переводящегося на русский язык как «дело» или «досье». В информатике под файлом понимают совокупность в некотором смысле однородных данных, характеризующих какой-либо процесс или некоторый объ-

ект. Мы будем иметь дело с программными файлами, состоящими из одной или нескольких именованных программ, и файлами данных.

Файлы обычно хранятся на тех или иных внешних носителях информации. Довольно часто, говоря о файле, имеют в виду не только определенные данные, но и соответствующую область носителя, на котором они размещены. Впрочем, из контекста всегда ясно, о чем идет речь, и к недоразумениям это не приводит.

Компоненты программного файла называют модулями или сегментами. Один из них считается основным. Он в первую очередь загружается в оперативную память и управляет всем процессом обработки информации. В частности, основной модуль по мере надобности проводит автоматическую подгрузку в память других сегментов файла. Именем файла считается имя его основного модуля. При небольших размерах программного файла разбивать его на модули нецелесообразно.

П Р И М Е Р 1. Программный файл из одного модуля

```
10 ' ФОРМУЛА  С И М П С О Н А
20 '-----------------------программа
30 COLOR 1,7,15:SCREEN 0
40 INPUT "нижний предел     ";A
50 INPUT "верхний предел    ";B
60 INPUT "количество точек ";M
70 CLS:LOCATE 8,8
80 PRINT "ПРОВОДЯТСЯ ВЫЧИСЛЕНИЯ"
90 S=0:H=(B-A)/M:S1=0:S2=0
100 FOR X=A+H/2 TO B-H/2 STEP H
110 GOSUB 220:S1=S1+Y
120 NEXT X
130 FOR X=A+H TO B-H STEP H
140 GOSUB 220:S2=S2+Y
150 NEXT X
160 X=A:GOSUB 220:S=S+Y
170 X=B:GOSUB 220:S=S+Y
180 S=(S1*4+S2*2+S)*H/6
```



```
190 CLS:LOCATE 6,8
200 PRINT "интеграл = ";S:END
210 '---------------подпрограмма
220 Y=(X^2)*SIN(X):RETURN
```

Нас будут интересовать два вида файлов данных: прямого доступа и последовательного доступа.

Компоненты файла прямого, или, по-другому, случайного доступа — это отдельные нумерованные группы значений, называемые записями. Запись может быть считана с внешнего носителя в буфер оперативной памяти и наоборот — сформирована в буфере и затем переписана во внешнюю память. Иными словами, буфер является своеобразным окном, через которое можно «рассматривать» файлы данных. Обращение к конкретной записи осуществляется прямым заданием ее номера.

Компоненты файла последовательного доступа называются *строками*. Каждая строка имеет специальную концевую метку и, как и запись, состоит из отдельных значений. Однако строки не нумерованы и обрабатывать их можно лишь последовательно друг за другом. Причем запись организуется от начала или конца файла, а чтение — только от его начала.

Заметим, что отдельные сегменты программных файлов — это, вообще говоря, специфические файлы данного последовательного доступа.

Оба рассмотренных вида файлов с успехом могут быть использованы в рамках расширенного Бейсика при работе с дисководом. В качестве внешних носителей в этом случае применяются магнитные диски. Последовательные файлы могут использоваться с любым, в том числе и нерасширенным, Бейсиком. Здесь устройством обмена является *магнитофон*, а внешним носителем — *магнитная лента*.

Следует иметь в виду, что организация работы с файлами прямого доступа в некоторых отношениях сложнее, чем с файлами последовательного доступа. Однако возможность непосредственного обращения к требуемой группе значений, предоставляемая файлами прямого доступа, позволяет считать их наиболее предпочтительными структурами хранения данных во внешней памяти.

4.2. Дисковые файлы

В главе «Файлы» данный раздел является доминирующим как по объему, так и по содержанию излагаемого материала. Он посвящен описанию команд и функций расширенного Бейсика, а также их использованию при работе с файлами.

Сервисные команды. В расширенном Бейсике предусмотрены группы команд и функций, с помощью которых организуется работа

отдельно с программными файлами, файлами данных прямого доступа и файлами данных последовательного доступа. Однако эта версия языка имеет и ряд средств, пригодных для работы с файлами любого типа. К ним относятся следующие команды и функции.

FILES (LFILES)	— команда вывода на экран (принтер) списка имен файлов.
DSKF	— функция, имеющая значением размер в байтах свободной части диска.
NAME... AS	— команда изменения имени файла на диске.
COPY... TO	— команда создания дубликата файла на диске.
KILL	— команда стирания файла на диске.

В последующих пунктах этого раздела обсуждается синтаксис и семантика перечисленных команд и функций, а также коротко затрагиваются вопросы, связанные с форматированием дисков и правилами образования имен файлов.

Форматирование диска. Для работы с 3,5-дюймовым дисководом MSX-компьютера используются двухсторонние диски MF2-DD, односторонние диски MF1-DD или их аналоги. На рисунке 24 показан общий вид такого диска, заключенного в пластмассовый кожух и имеющего переключатель защиты от записи и скользящую шторку для прикрытия окна, через которое организуется чтение-запись информации. Вставлять диск в щель (карман) дисковода и извлекать его оттуда можно лишь при включенном устройстве.

Разрешается подключать к компьютеру как одиночные, так и спаренные дисководы. В первом случае за устройством закрепляется переменное имя, которое в любой конкретный момент времени принимает одно из двух возможных значений А или В. Во втором случае основной дисковод получает имя А, а дополнительный — В.

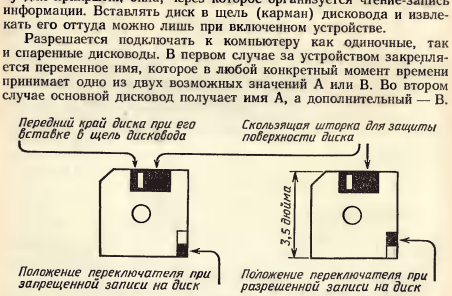


Рис. 24

Везде ниже, если это не оговорено специально, изложение ориентировано на использование одиночного дисковода, а слова диск, дискета и флоппи-диск рассматриваются как синонимы.

Для подготовки диска к работе с командами и функциями Бейсика необходимо осуществить специальную разметку его поверхности, называемую *форматированием* (форматизацией). Для этого диск вставляется в карман дисковода и выполняется команда CALL FORMAT. При вводе команды слово CALL (вызывать) можно заменить символом подчеркивания (_). Выполнение CALL FORMAT вызывает индикацию на экране запроса

DRIVE NAME? (A, B)

Ответом на него должен быть ввод одной из букв A или B в зависимости от имени устройства, в котором находится диск. На появившийся второй запрос-подсказку

1 — DOUBLE SIDES
2 — SINGLE SIDES
? —

пользователь обязан отреагировать вводом 1, если диск двухсторонний, или 2, если диск односторонний. Далее, по указанию

STRIKE A KEY WHEN READY

необходимо нажать любую клавишу (при появлении сообщения «INSERT DISKETTE FOR DRIVE...» требуется произвести нажатие еще одной клавиши, это приведет к смене имени устройства). После этого и начинается процесс форматирования, т. е. разметение на диске управляющих меток. При этом ранее имевшаяся там информация разрушается. Правильное завершение этого процесса вызывает индикацию следующего сообщения:

FORMAT COMPLETE
OK

Форматирование новых дисков обязательно!

Приведем некоторые характеристики диска MF2-DD.

1. Емкость при записи:
   а) всего — 720 Кбайт,
   б) для файлов пользователя — 713 Кбайт.
2. Плотность при записи — 8717 бит/дюйм.
3. Среднее время доступа — 95 мс.
4. Максимальное количество файлов — 112.

Имена файлов. При создании на диске любого нового файла ему необходимо присвоить определенное имя. Оно должно быть

уникальным, т. е. не совпадать ни с одним из уже имеющихся имен файлов, сформированных на данном диске ранее.

Имя служит для идентификации файлов и строится из последовательности символов алфавита Бейсика. При этом должны учитываться следующие обстоятельства:

а) Количество символов имени не может превосходить 11. Попытка сформировать имя большей длины ни к чему не приводит. Правые лишние знаки отбрасываются.

б) Соответствующие большие и малые латинские буквы, используемые в имени, равнозначны.

в) Соответствующие большие и малые буквы русского алфавита, используемые в имени, не равнозначны.

г) В имени не должно быть знаков:

; , + " = [ ] * \ / ?

д) Символ двоеточие (:) используется только при задании составных имен для разделения их на две части, первая из которых идентифицирует устройство, а вторая — файл. В этом случае при записи файла на диск фиксируется лишь вторая часть имени.

е) Нельзя задавать имя, состоящее из одних пробелов, начинающееся с пробела или содержащее неконцевые пробелы.

ж) Имя не должно начинаться с точки (.). В нем не может быть более одного такого символа и не более 8 знаков перед ним.

При задании имени «с точкой» ЭВМ интерпретирует ее в качестве символа девятой позиции. Тем самым имя разделяется на две зоны: основная часть (до точки) и расширение (после точки). Иногда расширение имени называют его индексной частью. Номер начальной позиции основной части — 1, а расширения — 10. Если пользователь фиксирует имя без точки с количеством символов, бо́льшим 8, то ЭВМ автоматически разбивает его на основную и индексную части, вставляя в требуемое место точку.

ПРИМЕР 2. Имена файлов

ЗАДАНИЕ	ИНТЕРПРЕТАЦИЯ
ГУСИ	ГУСИ
Р.Q53	Р .Q53
TIR.	TIR
#	#
ЛУЧ	ЛУЧ
ВАЛ.процент	ВАЛ .про
ПРОТИВОГАЗ	ПРОТИВОГ.АЗ
A:РОЗА	РОЗА

Справочная информация о файлах. Имеются довольно простые средства для получения списка имен файлов, размещенных на диске (FILES), размера его свободной части (DSKF) и некоторой другой справочной информации.

а) Вывод имен файлов осуществляется по команде

[L] FILES [i] (1)

где: FILES (файлы), L — служебные слова; i — имя файла.

При выполнении команды (1) без параметра i список имен всех файлов диска или индицируется на экране (L отсутствует), или распечатывается принтером (L есть). В зависимости от типа экрана и его ширины индикация имен организуется в 3, 2 или 1 колонку. Распечатка списка проводится в 1 колонку — каждому имени отводится одна строка. При отсутствии файлов на диске выдается сообщение

FILE NOT FOUND (файл не найден) (2)

Если в команде (1) параметр i указан и на диске имеется файл с этим именем, то данный факт подтверждается выводом единственного имени i. В противном случае выдается сообщение (2).

ПРИМЕР 3

```
1)    files              3) 100 FILES
2)    FILES "ИГРЫ"       4)    LFILES
```

Обратите внимание на возможность использования в основной части имени или его расширении символов маскирования * и ?

ПРИМЕР 4

```
1) FILES "*.DOT"  - индикация всех файлов
                    с расширением "DOT"
2) FILES"LUX.*"   - индикация всех файлов
                    с основной частью "LUX"
3) FILES"1*.*"    - индикация всех файлов,
                    имена которых начинаются
                    на единицу
4) FILES"*.?U?"   - индикация всех файлов,
                    второй символ расширения
                    у которых есть "U"
```

б) Функция DSKF (n), где: DSKF (*disk free*) (свободная часть диска) — служебное слово; n — арифметическое выражение, имеет значением размер в Кбайтах свободной части диска. Целая часть n_0 значения аргумента n может быть равна 0, 1 и 2. Она определяет имя, закрепленное за дисководом:

$n_0=0$ — текущее имя,
$n_0=1$ — A,
$n_0=2$ — B.

ПРИМЕР 5

```
1)    print DSKF(1)
2)    print DSKF(2)
3)    print DSKF(0)
4)    X=1:print DSKF(X)
5) 10 IF DSKF(0) > 300 THEN 1000
```

в) Для выяснения размера в байтах области диска, отведенной под произвольный файл i, можно воспользоваться строкой (см. «Файлы данных последовательного доступа»):

OPEN i FORINPUT AS#1:PRINT LOF(1)

г) Дополнительные сведения, относящиеся уже не к произвольным файлам, а лишь к файлам данных, можно получить с помощью функций LOF, LOC и EOF (см. «Файлы данных прямого доступа», «Файлы данных последовательного доступа»).

Изменение имен. Изменение имени программного файла или файла данных производится по команде

NAME i AS j (3)

где: NAME (имя), AS (как) — служебные слова; i — старое имя файла; j — новое имя файла.

При выполнении команды (3) прерывание происходит или при отсутствии на дискете файла с именем i (нечего переименовывать), или при наличии на ней файла с именем j (j зарезервировано). В противном случае файл i получает новое имя j.

При задании j допускается использование знака «?». Но при реальном формировании j на диске эти символы будут заменены на элементы из соответствующих позиций i.

ПРИМЕР 6

```
1)    NAME "алфавит" AS "АЛФ"
2)    NAME X$ AS "X.1"
3) 200 NAME "PQR.XYZ" AS "?.?11"
```


В последнем случае новое имя будет выглядеть так:

P ␣␣␣␣␣␣␣␣. X11.

Копирование файла. Для дублирования на дисках программных файлов и файлов данных применяется команда

COPY i TO j (4)

где: COPY (копировать), TO (в) — служебные слова; i — имя исходного файла; j — имя формируемого файла.

При выполнении команды (4) по файлу любого формата с именем i создается его копия j. Имена i и j не должны совпадать. В том случае, когда файл j уже существует на диске, наряду с копированием i в j проводится изменение до требуемых размеров файла j и свободной части диска.

Если i или j — файлы данных, то они не должны быть открыты.

ПРИМЕРЫ 7

```
1)    COPY "AL" TO "LA"
2)    COPY "X" TO "X.1"
3) 20 FOR J=2 TO 20
   30 COPY "U.1" TO "U."+MID$(STR$(J),2)
   40 NEXT
4)    COPY "A:ПРОБА" TO "B:ПРОБА"
```

В команде COPY в основной части имени или его расширении можно использовать знак *, заменяющий произвольную последовательность символов.

Копирование всех файлов с одного диска на другой при спаренных дисководах проводится по командам:

COPY "A:*.*" TO "B:*.*" (с A на B)
COPY "B:*.*" TO "A:*.*" (с B на A)

На одиночном дисководе эта и подобные ей процедуры, например

COPY "A:*.MSX" TO "B:*.MSX"

реализуются по соответствующим сообщениям на экране периодической сменой дисков в кармане.

Стирание файла. По команде

KILL i (5)

где: KILL (разрушать, убивать) — служебное слово; i — имя файла, происходит разрушение на диске произвольного файла с

именем i и увеличение на соответствующую величину размера свободной части диска. Фактическому разрушению (стиранию) подвергается лишь справочная информация о файле.

Если i — файл данных, то при выполнении команды (5) он не должен быть открыт.

ПРИМЕРЫ 8

```
1)      KILL "БОЧКА"
2)      KILL "R.T"
3)      X$="ЮЮЮ":KILL X$
4) 100 FOR K=1 TO 25
   110 KILL "ЦЕХ."+MID$(STR$(K),2)
   120 NEXT
5)      KILL "ABC.*"
```

В последнем примере удаляется справочная информация о всех файлах с основной частью имени «ABC» и произвольным расширением.

Программные файлы. Ниже описаны средства для перемещения программ из оперативной памяти на диск и в обратном направлении. Речь идет о командах SAVE, LOAD, RUN и MERGE.

1. SAVE — команда записи программ на диск.
2. LOAD, RUN — команды чтения программ с диска.
3. MERGE — команда подгрузки в память программы, записанной на диске в формате ASCII.

Запись программы на диск. По команде SAVE i [,A], где: SAVE (сохранить), A — служебные слова; i — имя файла, программа из оперативной памяти под именем i записывается на диск. В зависимости от наличия или отсутствия параметра A запись проводится соответственно или в формате ASCII, или в форме внутреннего представления. Если данная программа впоследствии будет подгружаться в память для слияния с другими модулями (MERGE), то ее необходимо записывать в формате ASCII.

При наличии на диске программного файла или файла данных с именем i и записи на него новой программы под тем же именем старый файл пропадает и происходит пересчет размера свободной части диска в соответствии с новыми размерами i.

Заметим, что формат ASCII приводит к увеличению в несколько раз времени записи программ из оперативной памяти на диск, а также их чтения с диска в память. К тому же для их размещения на диске требуется приблизительно на 30% больше места, чем для соответствующих программ во внутреннем представлении.

ПРИМЕР 9

```
1)      SAVE "ЯГОДА",A
2)      SAVE X$
3)      M$="PN":SAVE M$
4)      SAVE "Ф.И"
5) 10 SAVE "MICRO",A
6) 10 K=777
   20 SAVE "ПАЛАТА."
      +MID$(STR$(K),2)
```

Чтение программы с диска. Для считывания программ с диска в оперативную память используются команды LOAD и RUN. Остановимся на каждой из них в отдельности.

а) По команде

LOAD i [,R] (6)

где: LOAD (загрузить), R — служебные слова; i — имя файла, прежде всего закрываются все файлы и оперативная память очищается от программ и данных. Далее, программа, записанная на диске в машинных кодах или формате ASCII под именем i, загружается в оперативную память. При наличии параметра R после загрузки производится запуск программы на счет.

ПРИМЕР 10

```
1)      LOAD "LUX"
2)      LOAD "X.Y"
3)      Z$=MID$(A$,5,3):LOAD Z$
4)      LOAD "PR",R
5) 100 LOAD "TRACTOR",R
```

б) По команде

RUN i [, R]

где: RUN (запуск, прогон), R — служебные слова; i — имя файла, в оперативную память загружается и запускается на счет программа, записанная на диске под именем i в машинных кодах или формате ASCII. При отсутствии параметра R перед загрузкой i закрываются все открытые файлы и память очищается от программ и данных. При наличии в (6) параметра R из памяти удаляются программы и значения переменных. Однако файлы данных не закрываются.

ПРИМЕР 11

```
1)      RUN "ALPHA"
2)      RUN "Ведомость",R
3) 50 RUN "ТИП-А"
```

Слияние программ. Интересные возможности по «соединению» (слиянию) программ предоставляются командой

MERGE i (7)

где: MERGE (слияние) — служебное слово; i — имя файла, записанного на диске в формате ASCII. Выполнение команды (7) начинается с закрытия всех файлов и чистки оперативной памяти от данных. Далее производится слияние программы k, находящейся в памяти, с программным файлом, записанным на диске в формате ASCII под именем i. При этом если в программах k и i совпадают некоторые номера строк, то сохраняются лишь соответствующие строки загружаемой программы i.

ПРИМЕР 12

```
1)      MERGE "TTT"
2)      L$="ТЕХНИКА":MERGE L$
3) 100 MERGE "LIST.3"
```

Файлы данных прямого доступа. Как мы уже отмечали, файлы данных прямого (случайного) доступа представляют собой последовательности нумерованных групп значений, называемых записями. Для работы с этими файлами в Бейсике предусмотрены следующие команды и функции.

MAXFILES — команда резервирования в оперативной памяти буферов для формирования или считывания записей файлов.

VARPTR — функция, указывающая начальные адреса конкретных буферов в памяти.

OPEN — команда открытия файла данных и привязки к нему одного из объявленных буферов.

CLOSE — команда закрытия файлов.

FIELD — команда разбиения буфера файла на отдельные поля.

LSET, RSET — команды присваивания строковых значений буферным переменным.

MKI$, MKS$, MKD$ — функции преобразования числовых значений в строковые при их присваивании буферным переменным.

CVI, CVS, CVD — функции преобразования строковых значений переменных буфера в числовые.

PUT — команда обновления старых или формирования новых записей файла на диске (сброс записей из буфера на диск).

GET — команда считывания записей файла с диска в оперативную память.

LOC — функция, имеющая значением номер текущей записи файла.

LOF — функция, указывающая размер файла в байтах.

Ниже приведены две схемы, дающие наглядное представление о последовательности операций, которые должны быть выполнены при перемещениях записей файлов прямого доступа из оперативной памяти на диск и в обратном направлении.

I. *Общая схема формирования новых записей уже существующего или вновь создаваемого файла* β:

а) В оперативной памяти резервируется буфер (MAXFILES).

б) Открывается файл β (OPEN).

в) Проводится распределение поля буфера файла под значения переменных (FIELD).

г) В буфере формируется запись (LSET, RSET, MKI$, MKS$, MKD$ ...).

д) Производится сброс записей из буфера на диск с присваиванием ей некоторого номера (PUT).

Пункты г) и д) повторяются столько раз, сколько это требуется.

е) Файл β закрывается (CLOSE).

II. *Общая схема считывания записей из уже существующего файла.*

$а_1$) Должны быть выполнены пунктами а) — в) предыдущей схемы.

$б_1$) Производится считывание с диска в буфер оперативной памяти записи с конкретным номером (GET).

$в_1$) По записи из буфера в соответствии с распределением его поля формируются значения требуемых переменных (CVI, CVS, CVD, ...).

Пункты $б_1$ и $в_1$ повторяются столько раз, сколько это требуется.

$г_1$) Файл β закрывается (CLOSE).

В последующих пунктах проводится детальное описание средств Бейсика для реализации предложенных схем работы с файлами данных прямого доступа.

Контрольные буферы файлов. По команде MAXFILES=α, где: MAXFILES — служебное слово; α — арифметическое выражение ($0 \leqslant INT(\alpha) = \alpha_0 \leqslant 15$), резервируются (объявляются) α_0+1 участков оперативной памяти для последующего временного хранения записей конкретных файлов. Эти участки называются буферами, или, более полно, контрольными буферами файлов (File Control Buffers — FCB), и обозначаются так: FCB0, FCB1, ..., FCB α_0. Каждый FCB занимает 265 байт, из которых 9 предназначены для управляющей информации, а 256 — для данных.

Следует иметь в виду, что FCB0 и FCB1 объявлены по умолчанию и что FCB_0 используется многими командами (SAVE, LOAD, MERGE, RUN и т. п.).

При выполнении команды MAXFILES, кроме резервирования буферов, производится «чистка» значений переменных и закрытие всех ранее открытых файлов данных.

ПРИМЕР 13

```
1) 10 MAXFILES=5
2) 50 MAXFILES=X+Y
3)    MAXFILES=3
```

Функция

$$\text{VARPTR}\left(\left\{\begin{array}{c}\#\,\text{n}\\ \text{v}\end{array}\right\}\right) \qquad (8)$$

где: VARPTR (*pointer of variation* — указатель переменной) — служебное слово; n — арифметическое выражение, целая часть значения которого определяет номер FCB (от 0 до MAXFILES); v — идентификатор числовой или символьной переменной, указывает место расположения данных в оперативной памяти.

Рассмотрим возможные случаи.

1) VARPTR (# n)

Функция (8) имеет значением адрес X начального байта, с которого расположен в памяти FCBn (см. рис. 25).

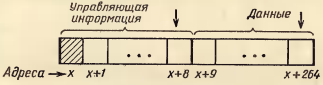


Рис. 25

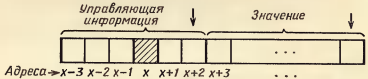


Рис. 26

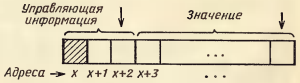


Рис. 27

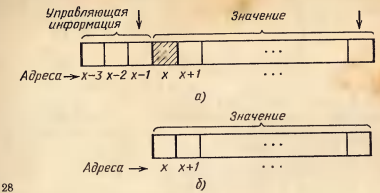


Рис. 28

2) VARPTR (v), v — строковая переменная.

В этом случае функция (8) имеет значением адрес X байта, находящегося на 3 позиции левее той, с которой располагается значение строковой переменной v. Схема размещения информации в памяти для простой строковой переменной изображена на рисунке 26, а для элемента строкового массива — на рисунке 27.

3) VARPTR (v), v — числовая переменная.

Здесь функция (8) имеет значением адрес того байта, начиная с которого располагается значение переменной v. Схема размещения информации в памяти для простой числовой переменной изображена на рисунке 28, *а*, а для элемента числового массива — на рисунке 28, *б*.

Функцию VARPTR можно использовать совместно с функциями PEEK (заглянуть) и POKE (впихнуть) соответственно для просмотра или изменения на свой страх и риск содержимого любой области оперативной памяти.

ПРИМЕР 14

```
1) 10 Y=VARPTR(#3)
2) 70 V=VARPTR(L$)
3)    PRINT VARPTR(X)
4) 10 A%(3)=99:                          run
   20 PRINT VARPTR(A%(3))                -27512
   30 PRINT PEEK(VARPTR(A%(3)))           99
5) 10 MAXFILES=2          40 Z=VARPTR(#2)
   20 X=VARPTR(#0)        50 PRINT X,Y-X
   30 Y=VARPTR(#1)        60 PRINT Y,Z-Y
```

Открытие файлов. Для работы с файлами данных, уже существующими на дисках или вновь организуемыми, необходимо произвести их открытие. Делается это командой

OPEN β AS [#] n [LEN=m] (9)

где: OPEN (открыть), AS (как), LEN (*length* — длина) — служебные слова; β — строковое выражение, задающее имя файла; n, m — арифметические выражения, целые части значений которых определяют соответственно номер контрольного буфера файла и длину его записей в байтах; # — необязательный символ, никоим образом не влияющий на выполнение команды (9).

Величина INT(n) должна лежать в диапазоне от 0 до MAXFILES. INT(m) может принимать значения от 1 до 256. По умолчанию m=256.

При выполнении команды OPEN файлу β назначается для работы контрольный буфер n. Кроме того, с диска в буфер заносится управляющая информация о файле (если β уже есть) или эта информация создается на диске (если β формируется заново). Вся эта процедура и называется *открытием файла*. Один файл β не может быть открыт сразу по нескольким контрольным буферам.

ПРИМЕР 15

```
1) 10 OPEN "ФАК" AS #1
2) 90 OPEN "клюква" AS #1 LEN=50
3) 10 MAXFILES=2
   20 OPEN "СТАРТ" AS #2
4) 10 MAXFILES=5
   20 OPEN "ФИНИШ" AS #3 LEN 100
```

Закрытие файлов. По команде

CLOSE [[#]n_1, [#]n_2, ..., [#]n_k]

где: CLOSE (закрыть) — служебное слово; n_1 n_2, ..., n_k — номера файловых контрольных буферов (FCB), организуется закрытие открытых по OPEN файлов данных, имеющих номера FCB:

n_1, n_2, ..., n_k.

Последние должны быть в пределах от 1 до 15. При отсутствии параметров в поле CLOSE закрываются все открытые на данный момент файлы данных. Закрытый файл становится недоступным для чтения записей из него в оперативную память (GET) и для формирования новых или обновления уже имеющихся записей (PUT).

Заметим, что закрытие файлов организуется также по командам END, CLEAR, LOAD, MAXFILES=, NEW и при любом изменении текста программы.

ПРИМЕР 16

```
CLOSE
CLOSE #3
CLOSE 1,5,7
```

Поля буфера файла. Как мы уже отмечали, под каждый контрольный буфер файла отводится 265 байт, 256 из которых непосредственно предназначены для временного хранения записей файла (см. MAXFILES, OPEN). Перед началом непосредственной работы с буфером необходимо произвести его разбиение на отдельные поля для формирования в них конкретных значений по команде

FIELD [#]n, m_1ASγ_1, m_2ASγ_2, ... m_kASγ_k (10)

где: FIELD (поле), AS (как) — служебные слова; n — арифметическое выражение, целая часть значения которого определяет номер буфера файла; m_s (s=1, 2, ..., k) — целые числа; γ_s (s=1, 2, ..., k) — строковые переменные.

При выполнении команды (10) производится задание формата записей. Это выражается в выделении каждому значению буферной переменной γ_s поля из m_s байт (s=1, 2, ..., k). Расположение этих полей показано на рисунке 29.

Отметим, что разрешается использовать несколько команд FIELD с разными форматами для одного и того же буфера n. Всегда действует то разбиение FCBn на поля, которое предложено последней выполненной командой FIELD.

ПРИМЕР 17

```
1) 10 FIELD #5, 10ASL1$, 20ASL2$, 15ASL3$
2) 100 MAXFILES=3
   110 OPEN "STUDENT" AS #2 LEN=40
   120 FIELD #2, 28ASM1$, 12ASM2$
   ...  ...  ...
   300 FIELD #2, 10ASM1$, 5ASM2$, 24ASK$
```

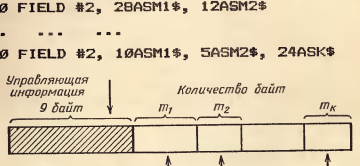


Рис. 29

Формирование строковых значений в буфере. Команда FIELD определяет формат записей файла и набор буферных переменных. Эти переменные не могут использоваться в предложениях LET, INPUT и LINE INPUT. При формировании в памяти конкретной записи с целью ее последующего сброса на диск значения переменным из FIELD присваиваются командами

LSET $\gamma=\beta$ (11)

RSET $\gamma=\beta$ (12)

где: LSET (*left set* — левое размещение), RSET (*right set* — правое размещение) — служебные слова: γ — буферная переменная; β — строковое выражение ($\beta \neq \gamma$).

Пусть переменной γ отведено в буфере поле из m байт. Выполнение команды (11) начинается с вычисления значения выражения β. Далее, если это необходимо, его длина приводится к величине m за счет отбрасывания лишних правых символов или добавления справа недостающего количества пробелов. Полученное значение присваивается γ и размещается в соответствующем поле буфера файла.

Выполнение команды (12) проводится почти так же, как и команды (11). Разница состоит лишь в том, что при выравнивании значения β до длины m возможные недостающие пробелы добавляются к нему не справа, а слева.

ПРИМЕР 18

```
10 OPEN "ef" AS #1 LEN=9
20 FIELD #1, 5ASM$, 4ASL$
30 INPUT R$:LSET M$=R$
40 INPUT R$:RSET L$=R$
50 PRINT M$
60 PRINT L$
```

```
run
? TOP
? ПАР
 TOP
  ПАР
run
? КРАТЕР
? НЕБОСВОД
КРАТЕ
НЕБО
```

Команды LSET и PSET можно использовать не только при работе с файлами.

ПРИМЕР 19

```
10 L$=SPACE$(20):Z$="ВОЛК"
20 RSET L$=Z$:PRINT L$
```


Формирование числовых значений в буфере. Буферные переменные являются строковыми. Пусть γ — такая переменная и ей командой FIELD в FCBn отведено поле длины m. Если требуется «упаковать» в γ числовое значение, то пользуются перечисленными ниже функциями.

а) MKI$ (α), где α — арифметическое выражение целого типа, m = 2; б) MKS$ (α), где α — арифметическое выражение одинарной точности, m = 4; в) MKD$ (α), где α — арифметическое выражение двойной точности, m = 8.

Каждая из этих функций преобразует значение арифметического выражения α к строковому типу. После этого командами LSET или RSET эти значения можно разместить в γ и соответствующем поле буфера. В рассматриваемом случае LSET и RSET выполняются идентично.

ПРИМЕР 20

```
10 MAXFILES=5
20 OPEN "LU" AS #4
30 FIELD #4, 2ASA$, 4ASB$, 8ASE$
50 INPUT X%:LSET A$=MKI$(X%)
70 INPUT Y!:LSET B$=MKS$(Y!)
90 INPUT Z#:LSET E$=MKD$(Z#)
100 PRINT A$;" ";B$;" ";E$
110 RSET A$=MKI$(X%)
120 RSET B$=MKS$(Y!)
130 RSET E$=MKD$(Z#)
140 PRINT A$;" ";B$;" ";E$
```

Получение числовых значений из буфера. Пусть в буфер с диска считана запись и в некотором ее поле, определенном в команде FIELD элементом mASγ, «упаковано» числовое значение. Обратная его распаковка, или, по-другому, выборка из γ этого значения реализуется одной из следующих функций.

CVI (γ). При m = 2 и выборке значения целого типа.
CVS (γ). При m = 4 и выборке значения одинарной точности.
CVD (γ). При m = 8 и выборке значения двойной точности.

ПРИМЕРЫ 21

```
1) 100 PRINT CVI(A$),CVS(B$),CVD(C$)
2) 10 OPEN "SL" AS #1 LEN=8
   20 FIELD #1, 8ASR$
```

```
30 INPUT X:RSET R$=MKD$(X)
40 PRINT R$,CVD(R$)
```

```
run
? 123
C0                    123
```

```
3) 10 INPUT X
   20 PRINT CVD(MKD$(X))
```

Сброс записей на диск. а) Для пересылки (сброса) сформированной в оперативной памяти записи на диск используется команда

PUT [#] n [, p] (13)

где: PUT (записать, разместить) — служебное слово; n, p — арифметические выражения, целые части значений которых задают соответственно номер контрольного буфера файла и номер записи.

До выполнения команды (13) в оперативной памяти должен быть объявлен контрольный буфер файла FCBn (MAXFILES=), открыт некоторый файл β, связанный с FCBn (OPEN), задан формат его записей (FIELD) и, наконец, в FCBn создана сама запись (LSET, RSET). После всей этой процедуры по команде (13) содержимое FCBn будет выведено на диск под номером p, или, как говорят, будет сформирована новая или обновлена старая запись p файла β.

Опишем выполнение (13) при отсутствии параметра p. Текущим значением p считается номер той записи, с которой работала последняя выполненная команда PUT или GET. При открытии файла p полагается равным 0. Далее, если в (13) p не указано, то берется его текущее значение, увеличенное на 1.

Напомним, что при создании записей числовые значения «упаковываются» в буферные переменные с помощью функций:

MKI$, MKS$ и MKD$.

ПРИМЕР 22

```
10 ' ОБЪЯВЛЕНИЕ FCB0, FCB1, FCB2 и FCB3
20 ' ОТКРЫТИЕ ФАЙЛА "РЫБА" С БУФЕРОМ #2
30 MAXFILES=3:OPEN "РЫБА" AS #2 LEN=13
40 ' ОПРЕДЕЛЕНИЕ ФОРМАТА ЗАПИСИ
50 FIELD #2, 8ASL$, 5ASM$
60 ' ФОРМИРОВАНИЕ ПЕРВЫХ 10 ЗАПИСЕЙ
70 FOR K=1 TO 10
80 INPUT "X$-8";X$:INPUT "Y$-5";Y$
90 LSET L$=X$:RSET M$=Y$:PUT #2, K
99 NEXT K:END
```


б) Пусть открыт некоторый файл данных β, связанный с буфером FCBn. Полезную информацию о β можно извлечь с помощью функций LOF и LOC.

Функция LOF (n) имеет значением размер файла β в байтах.

Функция LOC (n) имеет значением номер текущей записи файла β.

ПРИМЕРЫ 23

```
1)    PRINT LOC(1)
2) 10 X=LOF(3)
3) 99 PRINT LOF(2)
```

Считывание записей с диска. Для считывания с диска в оперативную память записей файла данных используется команда

GET [#] n [, p] (14)

где: GET (получить) — служебное слово; n, p — арифметические выражения, целые части значений которых задают соответственно номер контрольного буфера файла и номер записи.

До выполнения команды (14) необходимо объявить контрольный буфер FCBn (MAXFILES=), открыть некоторый файл данных β, связанный с FCBn (OPEN), и задать формат его записей (FIELD). После всей этой процедуры по команде (14) с диска в буфер #n будет считана запись с номером p. Фактически будут сформированы значения всех строковых переменных буфера (см. FIELD). Заметим, что числовые значения буферных переменных «распаковываются» с помощью функций: CVI, CVS и CVD.

Если в команде (14) p не указано, то берется текущее значение этого параметра, увеличенное на единицу (см. PUT).

В заключение этого пункта приведем пример, иллюстрирующий действие всех команд и функций при работе с дисковыми файлами данных прямого доступа.

ПРИМЕРЫ 24

```
10 ' ФАЙЛЫ ДАННЫХ ПРЯМОГО ДОСТУПА '---------
20 MAXFILES=5:PRINT VARPTR(#3)
30 OPEN "КАРАСЬ" AS #3 LEN=26
40 FIELD #3,8ASL$,4ASM$,2ASN$,5ASP$,7ASQ$
50 FOR K=1 TO 3:PRINT LOC(3)+1
60 INPUT "X дв. точность";X:LSET L$=MKD$(X)
70 INPUT "Y од. точность";Y:LSET M$=MKS$(Y)
75 INPUT "Z целое        ";Z:RSET N$=MKI$(Z)
```

```
80 INPUT "X$ - 5          ";X$:LSET P$=X$
90 INPUT "Y$ - 7          ";Y$:RSET Q$=Y$
 00 PUT #3, K
110 NEXT K
115 print STRING$(34,"-")
120 FOR K=3 TO 1 STEP -1
130 GET #3, K:PRINT LOC(3)
140 PRINT CVD(L$);CVS(M$);CVI(N$)
150 PRINT P$:PRINT Q$
160 NEXT:PRINT LOF(3):CLOSE #3 '-------------
run
-9380
 1
X дв. точность? 123E+45
Y од. точность? 123456
Z целое        ? 32000
X$ - 5         ? БОЧКА
Y$ - 7         ? АЛГЕБРА
 2
X дв. точность? 12345678901234
Y од. точность? 12
Z целое        ? 123
X$ - 5         ? ВОДА
Y$ - 7         ? МЕД
 3
X дв. точность? 1
Y од. точность? 156666666
Z целое        ? 22
X$ - 5         ? ПРОЛЕТАРИАТ
Y$ - 7         ? ДИСКОВОД
```



```
 3
 1  156667000  22
ПРОЛЕ
ДИСКОВО
 2
 12345678901234  12  123
ВОДА
    МЕД
 1
 1.23E+47  123456  32000
БОЧКА
АЛГЕБРА
 78
```

Файлы данных последовательного доступа. Файлы данных последовательного доступа состоят из отдельных групп значений, называемых логическими строками. Каждая строка имеет специальную концевую метку, а весь файл еще и метку EOF (конец файла). Строки рассматриваемых файлов не имеют номеров, и их можно считывать с диска в оперативную память или записывать из памяти на диск лишь последовательно друг за другом. Причем запись проводится от начала или конца файла, а чтение — только от его начала. Кроме того, любой конкретный файл открывается или только для чтения, или только для записи, но не для одновременного выполнения этих операций.

Для работы с файлами данных последовательного доступа в Бейсике предусмотрены следующие команды и функции.

MAXFILES — команда резервирования в оперативной памяти буферов для формирования или считывания строк файлов.

VARPTR
OPEN
CLOSE — см. «Файлы данных прямого доступа».

PRINT#
PRINT#USING — команды записи строк на диск.

INPUT#,
LINE INPUT# — команды считывания строк с диска в оперативную память.

INPUT$ — функция, имеющая значением фиксированное количество очередных символов файла.

EOF — булева функция, принимающая соответственно значения TRUE (истина) и FALSE (ложь) в зависимости от того, является ли очередной символ буфера меткой конца файла или нет.

LOF — функция, указывающая размер файла в байтах.

LOC — функция, имеющая значением текущее количество байт информации, считанных в буфер или записанных на диск.

Ниже приведены три схемы, определяющие последовательность операций, которые должны быть выполнены при перемещении строк файлов последовательного доступа из оперативной памяти на диск и в обратном направлении.

1. *Общая схема формирования строк на диске от начала файла* β.

а). В оперативной памяти резервируется буфер (MAXFILES).

б). Открывается новый или уже существующий файл β (OPEN с параметром OUTPUT).

в). Формируется требуемое количество строк файла (PRINT#, PRINT # USING). На диск из буферов они сбрасываются автоматически.

г). Файл β закрывается.

II. *Общая схема пополнения файла* β *строками с его конца.*

$а_1$). Выполняется пункт а) схемы 1.

$б_1$). Открывается уже существующий файл β (OPEN с параметром APPEND).

$в_1$). Выполняются пункты в) и г) схемы 1.

III. *Общая схема считывания строк с диска от начала файла* β.

$а_2$). Выполняется пункт а) схемы 1.

$б_2$). Открывается уже существующий файл β (OPEN с параметром INPUT).

$в_2$). Считывается требуемое количество строк файла (INPUT#, LINE INPUT#, INPUT$).

$г_2$). Файл β закрывается.

Более подробно об упомянутых выше командах и функциях рассказывается в следующих пунктах. При их описании через n обозначается арифметическое выражение, целая часть значения которого определяет номер контрольного буфера файла (FCBn).

Открытие и закрытие файлов. Перемещение строк последовательных файлов из оперативной памяти на диск и в обратном направлении производится через контрольные буферы файлов. По-

следние объявляются командой MAXFILES (см. «Файлы данных прямого доступа»). Для работы с файлом он должен быть открыт. Делается это командой

$$\text{OPEN } \beta \text{ FOR } \begin{Bmatrix} \text{INPUT} \\ \text{OUTPUT} \\ \text{APPEND} \end{Bmatrix} \text{ AS } [\#] \text{ n} \qquad (15)$$

где: OPEN (открыть), FOR (для), INPUT (ввод), OUTPUT (вывод), APPEND (присоединение), AS (как) — служебные слова; β — строковое выражение, задающее имя файла; n — номер контрольного буфера файла.

При выполнении команды (15) файлу β назначается для работы контрольный буфер n. Параметры INPUT, OUTPUT и APPEND указывают направление последующего движения информации относительно компьютера.

OPEN с параметром INPUT открывает файл для ввода строк с диска в буфер. Чтение будет организовано от начала файла.

OPEN с параметром OUTPUT открывает файл для вывода строк с буфера на диск. Запись осуществляется от начала файла.

OPEN с параметром APPEND открывает файл для вывода строк с буфера на диск. Запись будет производиться с конца файла.

ПРИМЕРЫ 25

```
1) 10 MAXFILES=2
   20 OPEN "МЕЛ" FORINPUT AS #2
2) 100 OPEN "RISS" FOROUTPUT AS #1
3) 200 OPEN "CAT" FORAPPEND AS #1
```

Закрытие файлов производится командой CLOSE.

Запись строк на диск. Для формирования строк последовательного файла используются команды PRINT# и PRINT#USING.

А. Команда PRINT# записывается в виде

$$\text{PRINT } \# \text{ n, s} \begin{Bmatrix} \sqcup \\ ; \\ , \end{Bmatrix} \qquad (16)$$

где: PRINT (печатать, вывести) — служебное слово; n — номер буфера файла; s — список арифметических и (или) строковых выражений: $\gamma_1, \gamma_2, \ldots, \gamma_k$.

При выполнении команды (16) значения γ_i ($i = 1,2, \ldots, k$) последовательно байт за байтом выводятся в буфер почти так же, как это делается командой PRINT при индикации γ_s на экране. Разница состоит лишь в трактовке зонного формата. Здесь он понимается так. Если между двумя выражениями из S в команде (16) разделителем является запятая, то при выводе в буфер соответствующих значений между ними вставляется 14 дополнительных пробелов.

В том случае, когда последним символом в (16) является пробел, реализация этой команды приводит к формированию полной строки вместе с меткой ее окончания. Иначе создается лишь часть строки, а завершается ее формирование или последующими командами PRINT#, или закрытием файла. В любом варианте длина строки вместе с концевой меткой не может превысить 256 байт, а данные выводятся на диск в формате ASCII и только тогда, когда заполняется буфер или закрывается файл.

Для экономного размещения данных на диске и избежания ошибок при их считывании целесообразно S в PRINT# записывать в виде

$$\gamma_1;\ ",";\ \gamma_2;\ ",";\ ...;\ ";\ \gamma_k \left\{ \begin{array}{c} \sqcup \\ ; \end{array} \right\}$$

В этом случае значения в создаваемой строке располагаются в плотном формате и отделяются друг от друга запятыми. Далее, если значение конкретного строкового выражения γ_i содержит символы "," и ";", то в качестве разделителей можно использовать кавычки, задаваемые с помощью кода CHR$ (34):

CHR$ (34) γ_i CHR$ (34)

Пример 26.

1) Открывается новый или ранее созданный файл «Щука» для вывода в него строк от начала и далее. Командой 50 в буфере формируется одна строка.

```
20 OPEN "ЩУКА" FOR OUTPUT AS #1
30 A=5:B$="ТРАВА":C=83.157
50 PRINT #1,A;",";B$;",";C
```

2) Открывается ранее созданный файл «Щука» для вывода в него строк от конца. Затем по команде 60 формируются очередные строки файла. По мере заполнения буфера данные выводятся на диск, а за последней строкой ставится метка EOF окончания файла.

```
20 OPEN "ЩУКА" FORAPPEND AS #1
30 INPUT "A= ";A
40 INPUT "B$= ";B$
50 INPUT "C= ";C
60 PRINT #1,A;",";B$;",";C
70 IF C<>0 THEN 30 ELSE CLOSE
```

3) Открывается новый или ранее созданный файл «Карась» для вывода в него строк от начала и далее. Двумя командами 40 и 50 формируется одна строка.


```
10 CLEAR 1000:MAXFILES=2
20 OPEN "КАРАСЬ" FOROUTPUT AS #2
30 A$="FLOPPYDISK":E=45:F=54.8
40 PRINT #2, A$;",";
50 PRINT #2, E;",";F
```

4) В файл «Карп» от его начала с пульта вводятся S элементов числового массива. Длина создаваемых строк, кроме, возможно, последней, равна 256 байт.

```
10 MAXFILES=5
20 OPEN "КАРП" FOROUTPUT AS #4
30 INPUT "количество элементов ";S
40 FOR K=1 TO S:INPUT M
50 PRINT #4,M;",";
60 NEXT K
```

5) При формировании строки в качестве разделителей используются кавычки, задаваемые кодом CHR$ (34).

```
20 OPEN "САЗАН" FOROUTPUT AS #1
30 R=123.789:K$="KRA,,N;W2"
40 I$=CHR$(34)
50 PRINT #1, R;",";I$;K$;I$
```

В. Команда PRINT#USING записывается в виде

PRINT#n,USING t, S (17)

где: PRINT, USING — служебные слова; t — строковое выражение, определяющее формат (шаблон) вывода; n, S — параметры, имеющие тот же смысл, что и в PRINT#.

При выполнении (17) значения выражений γ_i (i=1, 2, ..., k) выводятся в соответствии с форматом t в буфер точно так же, как это делается командой PRINTUSING при выводе γ_S на экран.

Пример 27

```
10 CLEAR 1000:MAXFILES=5:DIM M(200)
20 OPEN "ЛЕЩ" FOROUTPUT AS #4
30 INPUT "количество элементов ";S
40 FOR K=1 TO S:X=INT(RND(1)*3000)/99
50 PRINT #4, USING "###.##";X;
60 NEXT K:CLOSE #4
```

С. Пусть открыт файл β, связанный с буфером FCBn.

Как и для файлов прямого доступа, здесь LOF (n) имеет значением размер файла β в байтах.

Функция LOC (n) для файлов последовательного доступа имеет значением текущее количество байт информации, считанных в буфер или записанных на диск.

Чтение строк с диска. Для чтения строк последовательного файла с диска в оперативную память используются команды INPUT#, LINE INPUT# и функции INPUT$ и EOF.

А. По команде

LINE INPUT#n, γ

где: LINE (строка), INPUT (ввод) — служебные слова; n — номер буфера файла; γ — строковая переменная, считывается очередная строка файла β, связанного с буфером n, и присваивается переменной γ.

П Р И М Е Р Ы 28

```
1) 10 LINE INPUT #1, L$
2) 10 CLEAR 2000:MAXFILES=2
   20 OPEN "ЛЕС" FORINPUT AS #2
   30 LINE INPUT #2, M$
3) 10 CLEAR 1000:MAXFILES=5:DIM M(200)
   20 OPEN "LINP" FOROUTPUT AS #4
   30 INPUT "количество элементов ";S
   40 FOR K=1 TO S
   50 PRINT #4,K;",";:PRINT LOC(4);
   60 NEXT K:CLOSE #4
   70 OPEN "LINP" FORINPUT AS #4
   80 PRINT:PRINT LOF(4)
   90 LINE INPUT #4, M$
   100 PRINT LOC(4),:PRINT M$
   110 IF NOT EOF(4) THEN 90
   120 PRINT LOF(4):CLOSE #4
```

Обратите внимание на организацию завершения чтения строк в примере 28(3) по функции EOF (4).

В общем случае булева функция EOF (n) принимает значение TRUE (истина) или FALSE (ложь) в зависимости от того, является очередной символ буфера меткой конца файла или нет.

В. Команда INPUT# записывается в виде

INPUT#n, S (18)

где: INPUT (ввод) — служебное слово; n — номер буфера файла; S — список переменных.

Опишем, как происходит выполнение команды (18). Если очередная считываемая строка файла состоит из значений $\alpha_1, \alpha_2, ..., \alpha_k$, то в соответствующей команде INPUT# список S должен иметь вид: $\beta_1, \beta_2, ..., \beta_k$, а каждая переменная β_i обязана быть того же типа, что и α_i (i = 1, 2, ..., k). В этом случае реализация команды (18) приводит к присваиваниям переменным β_i значений α_i (i = 1, 2, ..., k).

П Р И М Е Р 29

```
10 CLEAR 1000:MAXFILES=5:DIM M(200)
20 OPEN "ШАР" FOROUTPUT AS #4
30 INPUT "количество элементов ";S
40 FOR K=1 TO S
50 X=RND(1):Y=RND(1)
60 X$=CHR$(INT(RND(1)*31)+224)
70 Y$=CHR$(INT(RND(1)*31)+224)
80 PRINT #4, X$+Y$;",";X;",";Y$
90 NEXT K:CLOSE #4
100 OPEN "ШАР" FORINPUT AS #4
110 INPUT #4, M$,A,B
120 PRINT M$;A;B
130 IF NOT EOF(4) THEN 110
140 CLOSE #4
run
количество элементов ? 5
ВЯ .59521943994623   .10658628050158
КЩ .73474759503023   .18426812909758
ЫО .63799556899423   .47041117641358
ОВ .18586284343823   .12951037284958
ТМ .02818381196223   .48775999680558
```

Заметим, что строка 110 в этом примере могла бы быть заменена на один из фрагментов:

```
а) 110 INPUT #4, M$
   111 INPUT #4, A,B
```

```
б) 110 INPUT #4, M$
   111 INPUT #4, A
   112 INPUT #4, B
```

С. Для чтения очередного фиксированного количества символов (байт) с файла без учета его разбиения на строки используется функция:

$$\text{INPUT\$ (m, \#n)} \qquad (19)$$

где: INPUT$ — служебное слово; m — арифметическое выражение, такое, что $m_0 = \text{INT (m)}$ лежит в пределах от 1 до 255; n — номер буфера файла.

В любой момент работы с файлом функция INPUT$ «читает» m_0 его очередных символов. Иными словами, последовательность этих символов становится ее значением. При этом метки окончания строк воспринимаются так же, как и любые другие знаки. Для полного уяснения семантики функции INPUT$ рекомендуется «прогнать» следующий пример при различных значениях параметров S и x.

```
П Р И М Е Р  30
10 CLEAR 1000:MAXFILES=5:DIM M(200)
20 OPEN "TOP" FOROUTPUT AS #4
30 INPUT "количество элементов   ";S
40 FOR K=1 TO S
50 PRINT #4, "L";",";"W";","
60 NEXT K:CLOSE #4
70 OPEN "TOP" FORINPUT AS #4
80 INPUT "длина значения в INPUT$";X
90 Y=(LOF(4)-1)\X:Z=(LOF(4)-1)MODX
100 IF Y=0 THEN 150
110 FOR K=1 TO Y
120 X$=INPUT$(X,#4):PRINT X$
130 NEXT K
140 IF Z=0 THEN 160
150 PRINT INPUT$(Z,#4)
160 CLOSE #4
```


Чтение программных файлов. Отдельные сегменты программных файлов — это специфические файлы данных последовательного доступа. Поэтому программы или их фрагменты можно загружать в память как данные. Ограничимся здесь двумя примерами.

```
П Р И М Е Р  31
10 ' ЧТЕНИЕ ПРОГРАММНОГО ФАЙЛА
20 OPEN "СИМПСОН" FORINPUT AS #1
30 LINE INPUT #1, R$:PRINT R$
40 FOR K=1 TO 1000:NEXT
50 IF NOT EOF(1) THEN 30
60 CLOSE
```

Выполнение этого фрагмента приводит к считыванию и индикации всех строк программы «Симпсон» (см. 4.1).

```
П Р И М Е Р  32
10 ' ЧТЕНИЕ ДИАПАЗОНА СТРОК
20 OPEN "СИМПСОН" FORINPUT AS #1
30 LINE INPUT #1, R$
40 X$=MID$(R$,1,2)
50 IF X$<"30" OR X$>"70" THEN 80
60 PRINT R$
70 FOR K=1 TO 1000:NEXT
80 IF NOT EOF(1) THEN 30
90 CLOSE
```

Выполнение данного фрагмента дает возможность «просмотреть» в той же самой программе «Симпсон» строки с номерами от 30 и до 70.

4.3. Файлы на магнитной ленте

Вывод программных файлов и файлов данных из оперативной памяти на магнитную ленту и обратное их считывание с ленты в память реализуется через магнитофон. Для идентификации файлам присваиваются имена, конструируемые из символов алфавита языка. Длина имени не может превосходить 6 (SAVE, CSATE) или 11 (BSAVE). Уникальность имен не обязательна.

Работа с магнитофоном может быть организована в рамках любой, в том числе и нерасширенной, версии Бейсика. Для этих целей предусмотрены следующие команды и функции.

SAVE	— команда записи в формате ASCII программ из оперативной памяти на магнитную ленту.
LOAD	— команда чтения (загрузки) в оперативную память программ, записанных на магнитной ленте в формате ASCII.
MERGE	— команда подгрузки (слияния) в память программ, записанных на ленте в формате ASCII.
RUN	— команда загрузки в память программ, записанных на ленте в формате ASCII, с автоматическим запуском их на счет.
CSAVE	— команда записи программы на Бейсике из оперативной памяти на ленту во внутреннем формате.
CLOAD	— команда чтения (верификации) в оперативную память Бейсик-программы, записанной на ленте во внутреннем формате.
BSAVE	— команда записи машинных кодов (программы или данные) из оперативной памяти на ленту.
BLOAD	— команда считывания модуля в машинных кодах (программы или данные) с ленты в оперативную память.
OPEN CLOSE MAXFILES EOF PRINT# PRINT#USING INPUT# LINEINPUT# INPUT$	— команды и функции для работы с файлами данных последовательного доступа (см. «Файлы данных последовательного доступа»).

Приступим к краткому описанию синтаксиса и семантики упомянутых выше средств организации файлов на магнитной ленте и работы с ними.

Запись и чтение программ в формате ASCII. В этом пункте описаны команды SAVE, LOAD, MERGE и RUN.

A. По команде SAVE α, где α — строковое выражение, программа из оперативной памяти записывается в формате ASCII на магнитную ленту. Имя программы формируется из первых 6 симво-

лов значения α. Оно может начинаться фрагментом CAS:, который в этом случае рассматривается как имя устройства и не учитывается при подсчете длины α.

ПРИМЕР 33

```
1) SAVE "ГРИБЫ"     2) SAVE "CAS:СОСНА"
```

B. Выполнение команды LOADα [, R] начинается с закрытия всех файлов и «чистки» оперативной памяти от программ и данных. Далее, программа, записанная на ленте в формате ASCII под именем, равным значению строкового выражения α, считывается в память. Наличие параметра R приводит к автоматическому запуску загруженной программы на счет.

Поиск конкретной программы на ленте сопровождается индикацией сообщений о всех «попавшихся» при этом других файлах, записанных в формате ASCII.

ПРИМЕР 34

```
LOAD "prim3"      Skip :prim2
Skip :prim1       Found:prim3
```

Сообщения выводятся на экран (*skip* — перескочить, *find* — находить).

Команда LOAD "CAS:" [, R] считывает в память первую встреченную на ленте в формате ASCII программу.

ПРИМЕР 35

```
1) LOAD "ФРУКТ"          3) LOAD "CAS:"
2) LOAD "CAS:LUX",R
```

C. При выполнении команды MERGE α закрываются все файлы и организуется чистка оперативной памяти от данных. Далее, программа, записанная на ленте в формате ASCII под именем, равным значению строкового выражения α, сливается с программой β, находящейся в памяти. При этом если в α и β совпадают некоторые номера строк, то сохраняются лишь соответствующие строки загружаемой программы α. Выдача сообщений на экране при поиске α здесь такая же, как и в случае с LOAD.

Команда MERGE "CAS:" подгружает в память первую встреченную на ленте в формате ASCII программу.

ПРИМЕР 36

```
1) MERGE "TTT"     3) MERGE "CAS:ALPHA"
2) MERGE "CAS:"
```

D. При выполнении команды RUN α прежде всего оперативная память очищается от программ и значений переменных. Открытые файлы не закрываются. Далее, программа, записанная на ленте в

формате ASCII под именем, равным значению строкового выражения α, загружается в память и автоматически запускается на счет. Выдача сообщений на экране при поиске α здесь такая же, как и в случае с LOAD.

ПРИМЕРЫ 37

```
1)      RUN "NET"      2) 110 RUN "CAS:LF"
3)      RUN X$ * X$ определено ранее
```

Запись и чтение программ во внутреннем представлении.

А. По команде CSAVE α, V, где α — строковое, а V — арифметическое выражение, программа из оперативной памяти во внутреннем представлении записывается на ленту под именем, равным значению α. V определяет скорость передачи информации (V=1—1200 бод, V=2—2400 бод). Впрочем, этот параметр может быть задан и в команде SCREEN.

ПРИМЕРЫ 38

```
1) CSAVE "ЛИСТ",2    2) CSAVE "ВЕРБА",1
3) A$="КЛЕН":CSAVE A$,2
```

В. По команде CLOAD α производится «чистка» оперативной памяти и загрузка в нее программы, записанной на ленте во внутреннем представлении под именем, равным значению строкового выражения α. Поиск α сопровождается индикацией на экране сообщений об именах всех «попавшихся» при этом файлов (см. LOAD).

ПРИМЕР 39

```
CLOAD "ТОРТ"

Skip :80008

Found:ТОРТ
```

С. По команде CLOAD ? α отыскивается программа, записанная на ленте во внутреннем представлении под именем, равным значению строкового выражения α, и побайтно сравнивается с программой, находящейся в данный момент в оперативной памяти. При идентичности сравниваемых текстов компьютер выходит в режим *Ok*. В противном случае получается сообщение

VERIFY ERROR (ошибка верификации)

ПРИМЕР 40

```
CLOAD? "СЛОВО"      FOUND:СЛОВО
Skip :1             Verify error
Skip :2
```


Команды BSAVE и BLOAD.

А. По команде BSAVE α, β_1, β_2, [, β_3], где: α — строковое выражение; β_1, β_2, β_3 — арифметические выражения, значения которых задают адреса, содержимое оперативной памяти от β_1 до β_2 в машинных кодах записывается на ленту под именем, равным значению строкового выражения α. β_3 определяет адрес, с которого программа при загрузке ее в память будет запускаться на счет. По умолчанию $\beta_3=\beta_1$.

ПРИМЕР 41

```
1) BSAVE "CAS:ГОСТ",&HF000,&HFEEE

2) BSAVE "CAS:"+P$,A,B,C
```

В. По команде BLOAD α [, R] [,σ], где: α — строковое выражение; R — параметр; σ — арифметическое выражение, в оперативную память загружается файл, записанный на ленте в машинных кодах под именем, равным значению α. При наличии параметра R прочитанная программа автоматически запускается на счет. Значение параметра σ должно быть целым. Оно определяет величину смещения адресов β_1, β_2 и β_3 загрузки и запуска (см. BSAVE). По умолчанию σ=0.

ПРИМЕР 42

```
1) BLOAD "CAS:DD"

2) BLOAD "CAS:LEN",&H1500

3) BLOAD "CAS:TEST",R

4) BLOAD "CAS:",R
```

Замечание. Команды BSAVE и BLOAD можно применять и для работы с произвольными дисковыми файлами.

Работа с файлами данных последовательного доступа. В разделе 4.2 перечислены команды и функции, используемые для работы с файлами данных последовательного доступа. Все эти средства подробно там описаны. Несколько иной синтаксис для данного случая имеет лишь команда OPEN — открытия файлов:

OPEN β FOR { INPUT / OUTPUT } AS #n

Здесь β должно начинаться с символов "CAS:" и, кроме того, отсутствует возможность, предоставляемая параметром APPEND для «наращивания» файлов.

```
П Р И М Е Р  43
10 OPEN "CAS:AL" FOROUTPUT AS #1
20 FOR K=1 TO 500:PRINT #1, K;
30 NEXT:CLOSE #1
40 STOP
50 CLEAR 1000
60 OPEN "CAS:AL" FORINPUT AS #1
70 LINE INPUT #1,M$:PRINT M$
80 IF NOT EOF(1) THEN 70
90 END
```

Здесь строки 10—30 создают файл данных на магнитной ленте. Для его считывания в память требуется запустить этот фрагмент со строки 50.

4.4. Вывод файлов данных на экраны и принтер

Текстовые экраны 0 и 1, графические экраны 2 и 3, а также принтер в рамках любой, в том числе и нерасширенной (недисковой), версии Бейсика могут быть объявлены устройствами вывода файлов последовательного доступа (см. 4.2). Открытие таких файлов осуществляется командой

$$\text{OPEN}\left\{\begin{matrix}\text{"CRT:"}\\ \text{"GRP:"}\\ \text{"LPT:"}\end{matrix}\right\}\text{FOR OUTPUT AS [\#] n} \qquad (1)$$

где: OPEN (открыть), FOR (для), OUTPUT (вывод), AS (как) — служебные слова; n — арифметическое выражение, целая часть значения которого определяет номер буфера файла; CRT, GRP, LPT — имена устройств.

При выполнении (1) в качестве устройства вывода файлов назначаются: текстовые экраны (CRT), графические экраны (GRP) или принтер (LPT). Вывод строк в любом случае осуществляется командами PRINT# и PRINT#USING (см. 4.2).

Пример 44.

1) Текст на графическом экране 2.

```
10 COLOR 1,11,7:SCREEN 2
20 OPEN "GRP:" FOROUTPUT AS #1
30 PSET (69,80)
40 PRINT #1, "РИС. 9.1 ЦВЕТА"
50 GOTO 50
```


2) Текст на графическом экране 3.

```
10 COLOR 1,11,7:SCREEN 3
20 OPEN "GRP:" FOROUTPUT AS #1
30 PSET (50,80)
40 PRINT #1, "БУКВЫ"
50 GOTO 50
```

3) Программа «777», записанная на диске в формате ASCII, выводится на графический экран 2.

```
10 COLOR 1,15,8:SCREEN 2
20 PSET (0,10),15:MAXFILES=2
30 OPEN "GRP:" FOROUTPUT AS #1
40 OPEN "777" FORINPUT AS #2
50 LINE INPUT #2, R$
60 PRINT #1, R$
70 FOR K=1 TO 1000:NEXT
80 IF NOT EOF(2) THEN 50
90 CLOSE:GOTO 90
```

4) Файл данных последовательного доступа, записанный на диске под именем «КАРП», выводится на принтер.

```
10 MAXFILES=2
20 OPEN "КАРП" FORINPUT AS #1
30 OPEN "LPT:" FOROUTPUT AS #2
40 INPUT #1, M:PRINT #2, M;
50 IF NOT EOF(1) THEN 40
```

4.5. Локальная вычислительная сеть

Между компьютерами «Ямаха», соединенными линиями передачи данных в локальную вычислительную сеть, возможен обмен информацией. Один из компьютеров сети получает статус «учительский» (головной), а остальные — «студенческий» (периферийный). Количество последних не должно превосходить 15. Взаимодействие между компьютерами сети сводится к нерегулярному обмену файлами между ними, а в остальном машины независимы. Поэтому рассматриваемую локальную сеть можно отнести к полностью распределенной системе.

Везде ниже при описании процедур передачи программ и данных по сети используются такие обозначения: *T* — учительский компьютер; *S* — студенческий компьютер.

Для идентификации за *T* закрепляется номер 0, а за S — номера: 1, 2, 3, ... Определить номер конкретного компьютера можно по команде-запросу CALL WHO.

Пересылка файлов от головного компьютера (*T*) к периферийным (*S*). При всех операциях, связанных с пересылкой программ и данных из *T* в *S*, информация пользователя в оперативной памяти учительского компьютера *T* не изменяется.

Загрузка программ из *T* в *S*.

А. На студенческих компьютерах *S*, в которые необходимо загрузить программу из *T*, нажатием клавиши F8 выполняем команду

LOAD "COM:" (клавиша F8)

Это приводит к чистке оперативной памяти соответствующих *S* и настраивает их на прием программы из *T*.

В. На учительском компьютере *T*, откуда пересылается программа, нажатием клавиши F7 выполняем команду

SAVE "COM:" (клавиша F7)

После этой процедуры происходит пересылка текущей программы из оперативной памяти компьютера *T* в те *S*, которые в данный момент находились в состоянии LOAD "COM:". Завершается пересылка выходом *T* и *S* в режим *Ok*.

Загрузка программы из *T* в *S* с запуском.

А1. На компьютерах *S*, в которые необходимо загрузить программу из *T*, нажатием клавиши F6 выполняем команду

RUN "COM:" (клавиша F6)

Это приводит к чистке оперативной памяти соответствующих *S* и настраивает их на прием программы из *T*.

В1. Выполняем пункт В (загрузка из *T* в *S*).

В результате этих действий происходит пересылка текущей программы из оперативной памяти компьютера *T* в те *S*, которые находятся в режиме RUN "COM:". Загруженные в *S* программы автоматически запускаются на счет.

Подгрузка программы из *T* в *S*.

А2. На всех студенческих компьютерах *S*, в которые необходимо «подгрузить» программу из *T*, нажатием клавиши F9 выполняем команду

MERGE "COM:" (клавиша F9)

В2. Выполняем пункт В (загрузка из *T* в *S*).

Эти действия приводят к «подгрузке» текущей программы α из *T* к соответствующим программам β тех *S*, которые находятся в режиме MERGE "COM:". Термин «подгрузка» означает такое слияние α и β, при котором в памяти *S* создается новая программа из α и тех строк β, номера которых не совпадают ни с одним из номеров строк α.

Селективная загрузка программ из *T* в *S*. Из учительского компьютера *T* можно пересылать программу α не только сразу во все *S*, находящиеся в состоянии RUN "COM:", но и выборочно в некоторые такие *S*. Подобная селективная загрузка реализуется разными способами. Опишем один из них.

А3. Пусть пересылаемая из *T* программа α начинается со строки n ($n \geqslant 8$). «Надстроим» ее вспомогательным фрагментом (строки 1—7 примера 45), где в команде DATA укажем сначала общее количество *S*, а затем номера тех *S*, которые должны получить α.

```
П Р И М Е Р  45
1 DATA 3,1,3,4
2 READ X
3 FOR K=1 TO X:READ Y
4 IF Y=INP(2)AND15 THEN 7
5 NEXT
6 RUN "COM:"
7 DELETE 1-7
10 'P R O G R A M M A
20 ' ...   ...   ...
```

В3. Выполняем на *T* команду

SAVE "COM:" (клавиша F7)

После этой процедуры происходит следующее. Программа α пересылается во все *S*, находящиеся в состоянии RUN "COM:", и запускается на счет. Далее, если номер конкретного компьютера *S* присутствует в списке DATA, то фрагмент 1—7 организует чистку самого себя и выводит *S* в режим *Ok*. В противном случае, то есть когда номер *S* отсутствует в DATA, этот фрагмент производит чистку всей памяти, переводя *S* в состояние RUN "COM:". В результате этих операций программа α оказывается загруженной только в требуемые компьютеры.

Пересылка данных из *T* в *S*. Программам β, запущенным на счет на студенческих компьютерах *S*, можно передавать из *T* значения простых переменных и элементов массивов. Проиллюстрируем на примерах один из возможных способов подобной пересылки данных.

А4. На всех студенческих компьютерах *S*, куда будут передаваться данные из *T*, должна быть запущена подпрограмма (фрагмент), в которой:

а) объявлены массивы для приема значений;

б) открыт файл данных последовательного доступа "COM:" с параметром INPUT;

в) организовано чтение строк из *T* в *S*.

```
П Р И М Е Р  46

...   ...

100 GOSUB 1000

...   ...

1000 DIM A(30),B$(30)

1010 OPEN "COM:" FORINPUT AS #1

1020 FOR K=1 TO 30

1030 INPUT #1, A(K),B$(K)

1040 PRINT A(K),B$(K)

1050 NEXT:CLOSE #1:RETURN
```

В4. На учительском компьютере *T*, откуда будут передаваться данные в *S*, должен быть запущен фрагмент, в котором:

а) открыт файл данных последовательного доступа "COM:" с параметром OUTPUT;

б) организована пересылка строк из *T* в *S*.

```
П Р И М Е Р  47 (соответствует примеру 46)

2000 OPEN "COM:" FOROUTPUT AS #1

2010 FOR K=1 TO 30

2020 PRINT #1, K+100;",";"ABS"

2030 PRINT K+100;"ABS"

2040 NEXT:END
```

Эти действия приводят к загрузке данных из *T* в те *S*, которые находились в состоянии INPUT# (LINE INPUT#, INPUT$). После завершения этой процедуры программы в *S* продолжат свою работу.

Пересылка программы и данных из *T* в *S*. Ограничимся здесь конкретными примерами. Пусть требуется переслать в *S* из оперативной памяти *T* программу α и с диска — элементы массива A_k (k=1, 2, ..., 500), организованные там в виде файла последовательного доступа с именем "MAS".

А5. Будем считать, что наименьший номер строк программы — 10. «Надстроим» ее следующим вспомогательным фрагментом (строки 1—9 примера 48).

```
П Р И М Е Р  48

1 DIM A(500):OPEN "COM:" FORINPUT AS #1

2 FOR K=1 TO 500:INPUT #1, A(K)

3 PRINT A(K);

4 NEXT:CLOSE #1:GOTO 10

5 MAXFILES=2:OPEN "MAS" FORINPUT AS #2

6 OPEN "COM:" FOROUTPUT AS #1

7 FOR K=1 TO 500:INPUT #2, M:PRINT #1, M;

8 PRINT M;

9 NEXT:END

10 '     П Р О Г Р А М М А
```

В5. На всех студенческих компьютерах *S*, куда будут пересылаться программа и данные, выполним команду

RUN "COM:" (клавиша F6)

Это приводит к чистке оперативной памяти соответствующих *S* и настраивает их на прием программы из *T*.

С5. На учительском компьютере *T* выполняем команду

SAVE "COM:" (клавиша F7)

В результате реализации этих действий в те *S*, которые находились в состоянии RUN "COM:", загружается и запускается на счет программа, а *T* выходит в режим *Ok*.

D5. Для пересылки данных с диска в *S* выполняем на *T* команду RUN 5.

Набор RUN 5 необходимо осуществлять по отдельным символам (RUN на спецклавише реализован вместе с кодом запуска CHR$ (13)).

После «перекачки» данных программа *S* будет продолжать работу со строки 10.

Заметим, что предложенный способ организации передачи программы и данных удобен тем, что сводит к минимуму действия, которые должны быть осуществлены за пультом студенческих компьютеров. Фактически на них требуется лишь нажать клавиши F6.

Однако следует иметь в виду, что в программах, загруженных в *S*, будет содержаться фрагмент из строк 1—9, являющийся для них «мусором».

Режим пояснений. С клавиатуры учительского компьютера *T* на экранах *S* можно набирать различного рода сообщения, пояснения, тексты программ и т. д. Делается это так.

А6. На студенческих компьютерах *S*, с которыми будет проводиться общение, набираем команду (клавиша F10)

__comterm (1)

(в записи (1) символ "—" (подчеркивание) является обозначением для слова *call* (вызывать).

В6. На *T* выполняем ту же самую команду (1).

После этого нажатие на клавиатуре *T* любой клавиши приводит к индикации соответствующего символа или выполнению надлежащего действия на экранах всех *S*, находящихся в режиме "_comterm". При этом собственный экран *T* оказывается заблокированным. Поэтому следить за правильностью сообщений учителю приходится по экрану какого-либо соседнего компьютера *S*.

Отметим следующее немаловажное обстоятельство. Вся информация из *T* в *S* поступает в так называемую видеопамять, а в оперативной памяти не запоминается.

Пересылка файлов от периферийного компьютера к головному. Все операции по пересылке программ и данных из конкретного студенческого компьютера *S* в учительский *T* осуществляются точно так же, как и в обратном направлении. Лишь по отношению к действиям пользователей *T* и *S* меняются ролями. При этом следует учитывать, что при любых пересылках из *S* в *T* содержимое оперативной памяти *S* не меняется, а попытки организовать передачу информации одновременно из нескольких *S* в *T* приводят к сбоям.

В заключение отметим, что за движением информации по сети можно «наблюдать» на экране дисплея любого компьютера *S*. Для этого его необходимо перевести в соответствующий режим, выполнив команду

__comterm (клавиша F10)

5. БЕЙСИК MSX-2

По сравнению с Бейсик-системой MSX-1 ее дальнейшее развитие, Бейсик-система MSX-2 (версия 2.1), реализованная на компьютерах «Ямаха» MSX-2, предоставляет пользователю ряд интересных дополнительных возможностей. Стало больше доступных режимов видеопроцессора (экраны 0—8), расширен объем ОЗУ и за счет этого организована работа с виртуальным диском, появились средства для

начальной инициализации ряда параметров, изменились принципы функционирования локальной вычислительной сети. На этих и других вопросах и будет сосредоточено внимание в этом разделе.

5.1. Экраны

Основные характеристики. Каждый из 9 имеющихся режимов видеопроцессора будем именовать «экраном» с добавлением после этого цифры от 0 до 8. Основные характеристики экранов приведены в таблицах 24—25. Там использованы такие обозначения: Т — текст (текстовой), Г — графика (графический), S — спрайты, SC — цветные спрайты.

При включении компьютера по умолчанию устанавливается один из текстовых режимов SCREEN 0 или SCREEN 1. Для его изменения необходимо в непосредственном счете или программном режиме выполнить оператор SCREEN N, где значениями арифметического выражения N должны быть числа 0, 1, 2, ..., 8.

Таблица 24

Основные характеристики экранов

№ экрана	MSX-1 MSX-2	Тип экрана	Разрешающая способность	Возможности вывода
0	1, 2	Т	240×192	Т
1	1, 2	Т	256×192	Т S
2	1, 2	Г	256×192	Т, Г, S
3	1, 2	Г	64×48	Т, Г, S
4	2	Г	256×192	Т, Г, S, SC
5	2	Г	256×212	Т, Г, S, SC
6	2	Г	512×212	Т, Г, S, SC
7	2	Г	512×212	Т, Г, S, SC
8	2	Г	256×212	Т, Г, S, SC

Таблица 25

№ экрана	Номер цвета			Палитра
	Изображение	Фон	Бордюр	
0	0—15	0—15	—	+
1	0—15	0—15	0—15	+
2	0—15	0—15 (CLS)	0—15	+
3	0—15	0—15 (CLS)	0—15	+
4	0—15	0—15 (CLS)	0—15	+
5	0—15	0—15 (CLS)	0—15	+
6	0—31	0—31 (CLS)	0—31	+ (4)
7	0—15	0—15 (CLS)	0—15	+
8	0—255	0—255 (CLS)	0—255	—

Таблица 26

№ экрана	Матрица символов	Количество символов в строке	Многоцветные тексты	Стирание символов пробелом	Номер страницы
0	6×8	40, 80	—	+	—
1	8×8	32	+	+	—
2	8×8	32	+	—	—
3	8×8	8	+	—	—
4	8×8	32	+	—	—
5	8×8	32	+	+	0, 1, 2, 3
6	8×8	64	+	+	0, 1, 2, 3
7	8×8	64	+	+	0, 1
8	8×8	32	+	+	0, 1

Цвет изображения (И), фона (Ф) и бордюра (Б) задается оператором

COLOR И, Ф, Б

Допустимые значения выражений И, Ф и Б приведены в колонках 2—4 таблицы 25. Из них видно, что цвет бордюра для экрана 0 не определяется. Он всегда совпадает с цветом фона. Далее, начиная с экрана 2 и выше, цвет фона непосредственно по оператору COLOR И, Ф, Б не изменяется. Это происходит лишь после переназначения экрана или выполнения оператора CLS. Для графических экранов 2—8 вывод текстов возможен только после назначения "GRP:" в качестве устройства вывода. В примере 1 предполагается, что вначале цвета установлены стандартным образом по COLOR 15, 4, 4.

ПРИМЕР 1

```
10 SCREEN 5:COLOR 15,12,4
20 OPEN "GRP:" AS #1
30 PSET (70,80),4:PRINT#1,"COLOR до CLS"
40 R$=INPUT$(1):CLS
50 PSET (58,80),12:PRINT#1,"COLOR после CLS"
```

Отметим также следующие обстоятельства.

1. В MSX-2 для экрана 0 максимальная длина строки, назначаемая оператором WIDTH, равна 80.

2. Для экрана 1 возможен вывод спрайтов, но отсутствует графика.

3. Для экрана 3 «точки» задаются блоками 4×4. Этим и объясняется его малая разрешающая способность.

4. Для экранов 2—8 вывод текстовой информации возможен с любой позиции графического курсора.

5. Для экранов 2—4 нельзя «чистить» текст пробелами.

6. Цветные спрайты реализованы только на экранах 4—8.

7. Для экранов 1—3 и 4—8 на одной горизонтальной линии могут находиться соответственно до 4 и до 8 спрайтов одновременно.

8. Экран 6 допускает всего лишь 4 различных цвета.

9. С экранами 5—8 возможен многостраничный режим работы.

10. Для вывода многоцветных рисунков наиболее удобным, по-видимому, является экран 7.

Палитра. В RGB-мониторе[1] зафиксировано 16 различных цветов, пронумерованных от 0 до 15 и представляющих собой определенные пропорции красного, синего и зеленого тонов (см. табл. 27). Эти пропорции могут быть изменены. Поэтому нет смысла говорить о номере конкретного цвета, а лучше использовать термин «номер палитры». В компьютерах MSX-2 изменение палитры реализуется простыми средствами. Делается это по оператору COLOR= (код, R, G, B), где арифметические выражения в скобках имеют смысл

1. код — номер изменяемого цвета (0—15);

2. R, G, B — пропорции красного, зеленого и синего цветов (0—7).

Таблица 27

Код	Цвет	Красный	Зеленый	Синий	Код	Цвет	Красный	Зеленый	Синий
0	прозрачный	0	0	0	8	красный	7	1	1
1	черный	0	0	0	9	светло-красный	7	3	3
2	зеленый	1	6	1	10	темно-желтый	6	6	1
3	светло-зеленый	3	7	3	11	светло-желтый	6	6	4
4	темно-синий	1	1	7	12	темно-зеленый	1	4	1
5	синий	2	3	7	13	сиреневый	6	2	5
6	темно-красный	5	1	1	14	серый	5	5	5
7	голубой	2	6	7	15	белый	7	7	7

В примере 2 при выполнении приведенного фрагмента программы последовательно получаем чистые синий, зеленый и красный цвета.

Оператор SCREEN, кроме чистки экрана, устанавливает па-

ПРИМЕР 2

```
1 ' НАЖИМАЙТЕ     ЛЮБУЮ КЛАВИШУ
2 ' ------------------------------
10 SCREEN 7:COLOR 15,4,4:CLS:GOSUB 50
20 COLOR=(4,0,0,4):GOSUB 50
30 COLOR=(4,0,4,0):GOSUB 50
40 COLOR=(4,4,0,0):GOSUB 50:GOTO 10
50 R$=INPUT$(1):RETURN
```

[1] R — RED (красный), G — GREEN (зеленый), B — BLUE (синий).

литры в исходное состояние (см. табл. 27). Этот же эффект, но без стирания изображения, достигается любым из операторов COLOR или COLOR=NEW.

Замечание. Если изменение палитры реализуется непосредственно в видеопамяти с помощью VPOKE, то на изображение это не повлияет до тех пор, пока не будет осуществлено считывание таблицы палитр оператором COLOR=RESTORE.

Отметим также имеющуюся для экрана 1 причудливую возможность выводить в различных цветах определенные части фона и отдельные символы. Для этого необходимо непосредственно воздействовать на таблицу цветов видеопамяти оператором VPOKE. Расположена эта таблица по адресам &H2000 — &H201F. Каждый ее байт кодирует цвета изображения (старшие 4 бита) и фона (младшие 4 бита) ровно для 8 последовательных символов. Точнее, по адресу &H2000+A (A=0, 1, ..., 31) кодируются цвета символов с CHR$-кодами от 8·A до 8·A+7. В примере 3 приводится инвертирование цвета для символов: 0, 1, 2, 3, 4, 5, 6, 7.

ПРИМЕР 3

```
10 SCREEN 1:COLOR 15,4
20 VPOKE &H2000+6,&H4F:GOSUB 50:R$=INPUT$(1)
30 VPOKE &H2000+6,&HF4:GOSUB 50:R$=INPUT$(1)
40 GOTO 20
50 LOCATE 7,10:PRINT"ТАРИФ-01234567":RETURN
```

В примере 4 показан вывод на экран всех знаков алфавита и случайное изменение цвета у отдельных групп символов.

ПРИМЕР 4

```
1 ' НАЖИМАЙТЕ      ЛЮБУЮ КЛАВИШУ
2 '-----------------------------------------
10 SCREEN 1:COLOR 15,4,4:X=RND(-TIME)
20 '                вывод символов псевдографики
30 FOR S=64 TO95:PRINT CHR$(1)+CHR$(S);:NEXT
40 '                        вывод символов алфавита
50 FOR S=32 TO 255:PRINT CHR$(S);:NEXT
60 DEFFNR(U)=INT(RND(1)*U)
70 X=FNR(32):C1=FNR(16):C2=FNR(16)
80 VPOKE &H2000+X,16*C1+C2
90 R$=INPUT$(1):GOTO 70
```


Из сказанного ранее вытекает, в частности, что возможен вывод на экран курсора в цвете, отличном от цвета изображения.

ПРИМЕР 5

```
10 SCREEN 1:COLOR 15,4,4:X=RND(-TIME)
20 LOCATE 3,10:PRINT "ЦВЕТ"
30 DEFFNR(U)=INT(RND(1)*U)
40 C1=FNR(16):C2=FNR(16)
50 VPOKE &H201F,16*C1+C2
60 LOCATE 12,10:R$=INPUT$(1):GOTO 40
```

Несмотря на кажущуюся экзотичность рассмотренных приемов изменения цвета, для экрана 1 именно они и подобные им трюки позволяют создавать весьма эффективные игровые программы.

Цветные спрайты. При рассмотрении простейших графических динамических объектов, называемых спрайтами, отметим, что каждый из них может быть раскрашен, но только в один конкретный цвет. Поэтому многоцветная мультипликация оказывалась возможной лишь при наложении спрайтов (спрайты с меньшими номерами перекрывают спрайты с бо́льшими номерами).

На экранах 4—8 допускаются и цветные спрайты. При этом шаблон спрайта формируется обычным образом (SPRITE$ (n) =), вывод его на экран также не претерпевает изменений (PUT SPRITE). Однако дополнительный оператор

COLOR SPRITE$ (n) = α (1)

позволяет раскрашивать отдельные горизонтальные линии выведенного спрайта, смещать эти линии влево на 32 точки и отменять назначенное прерывание при столкновении данного спрайта с другими.

В операторе (1): n — номер спрайта, α — строковое выражение, значение которого состоит из 8 или 16 символов.

При выполнении этого оператора двоичная структура кода ASCII каждого символа α действует на спрайтовой линии в соответствии со значениями своих отдельных битов. Объясним, как это происходит для конкретного «символа» — линии.

а) Биты 0—3 (четыре левых). Эти биты задают код палитры (0—15), присваиваемой соответствующей линии.

б) Бит 4. Не используется.

в) Бит 5. Если бит равен 1, то отменяется прерывание при пересечении данного спрайта с другими.

) Бит 6. Если бит равен 1, то происходит следующее:
отменяется прерывание при пересечении данного спрайта с другими;
при наложении спрайтов с цветами C1 и C2 их пересечение принимает цвет C1 OR C2;
линии, не затронутые пересечением, исчезают.

д) Бит 7. Если бит равен 1, то соответствующая линия отображается на 32 точке левее абсциссы в PUT SPRITE.

Пример 6 иллюстрирует раскрашивание полос спрайта.

```
П Р И М Е Р  6
10 SCREEN 4,1:COLOR 15,4,4:CLS
20 SPRITE$(0)=STRING$(8,255)
30 PUT SPRITE 0,(118,86),15,0
40 R$=INPUT$(1):GOSUB 80
50 R$="":FOR S=1 TO 8:R$=R$+A$(S):NEXT
60 COLOR SPRITE$(0)=R$
70 GOTO 70
80 A$(1)= CHR$(&B0000000001)
90 A$(2)= CHR$(&B0000000011)
100 A$(3)=CHR$(&B0000001000)
110 A$(4)=CHR$(&B0000011111)
120 A$(5)=CHR$(&B0000001111)
130 A$(6)=CHR$(&B0000001011)
140 A$(7)=CHR$(&B0000011101)
150 A$(8)=CHR$(&B0000011100)
160 RETURN
```

В этом примере по операторам строк 10—30 проводится формирование шаблона спрайта и его вывод на экран в виде белого квадрата на голубом фоне. Нажатие на любую клавишу организует обращение к подпрограмме инициализации массива строковых переменных. Далее, в строке 50 формируется шаблон R$ и, наконец, по COLOR SPRITE$(0) в соответствии с предписанием R$ все линии спрайта раскрашиваются в различные цвета. Например, линия 2 получает цвет «зеленый» потому, что

A$ (2) = CHR$ (&B00000011)

В примере 7 показано смещение полос спрайта. (Рассматривается программа примера 6 с измененной подпрограммой.)

```
П Р И М Е Р  7
80  A$(1)=CHR$(&B1000010011)
90  A$(2)=CHR$(&B0000001000)
100 A$(3)=CHR$(&B1000000111)
110 A$(4)=CHR$(&B0000010000)
120 A$(5)=CHR$(&B1000000010)
130 A$(6)=CHR$(&B0000001000)
140 A$(7)=CHR$(&B1000000001)
150 A$(8)=CHR$(&B0000010000)
160 RETURN
```

В этом случае спрайт как бы расщепляется на отдельные полосы так, что часть линий смещается на 32 точки влево и перекрашивается. Например, линия 7 принимает цвет «черный» и смещается влево потому, что

A$(7) = CHR$(&B10000001)

В примере 8 иллюстрируется наложение спрайтов с изменением цвета.

```
П Р И М Е Р  8
10 SCREEN 4,3:COLOR 15,4,4:CLS
30 SPRITE$(0)=STRING$(32,255)
40 PUT SPRITE 0,(118,86),2,0
50 PUT SPRITE 1,(138,90),8,0
60 T$=INPUT$(1)
70 COLOR SPRITE$(1)=STRING$(32,&B0100010000)
80 GOTO 80
```

По этому фрагменту программы операторы строк 40 и 50 выводят два спрайта-квадратика соответственно зеленого — 2 и красного — 8 цветов. Они имеют непустое пересечение и немного смещены относительно друг друга по вертикали. Поскольку зеле-

ный спрайт имеет экранный номер бо́льший, чем красный спрайт, то их общая часть также оказывается зеленой. Учитывая, что 2OR8=10, нажатие на любую клавишу приводит к тому, что пересечение спрайтов становится желтым. При этом линии, не затронутые пересечением, сливаются с фоном.

В примере 9 показано движение спрайта.

```
П Р И М Е Р  9

10 SCREEN 4,1:COLOR 15,4,4:CLS
20 SPRITE$(0)=STRING$(8,255)
30 PUT SPRITE 0,(0,86),15,0
40 Q$=CHR$(&B00001000):T$=CHR$(&B00001111)
50 COLOR SPRITE$(0)=Q$+Q$+T$+Q$+Q$+T$+Q$+Q$
60 FOR S=0 TO 256
70 PUT SPRITE 0,(S,86),,0
80 NEXT:GOTO 60
```

Здесь демонстрируется движение цветного спрайта. Поскольку управляющим параметром взята абсцисса точки в PUT SPRITE, то движение осуществляется по горизонтальной линии.

В случае раскраски спрайта сразу по всем линиям можно использовать оператор

COLOR SPRITE (n)=β (2)

Здесь: n — номер спрайта, β — арифметическое выражение со значениями от 0 до 127.

При выполнении (2) двоичная структура числа β, дополненная до 8 бит слева нулями, действует на каждой спрайтовой линии так же, как один символ на конкретной линии в (1). В примере 10 иллюстрируется одновременная раскраска всех линий.

```
П Р И М Е Р  10

10 SCREEN 4,3:COLOR 15,4,4:CLS
20 SPRITE$(0)=STRING$(32,255)
30 PUT SPRITE 0,(140,80),2,0
40 PUT SPRITE 2,(50,126),8,0
50 PUT SPRITE 3,(70,136),8,0
60 COLOR SPRITE(3)=71
70 GOTO 70
```


Отметим следующий факт. Поскольку в (2) седьмой старший бит структуры, воздействующей на спрайт, всегда равен нулю, то смещение полос по этому оператору невозможно.

Пример 11 показывает отмену прерывания по столкновению спрайтов.

```
П Р И М Е Р  11

10 SCREEN 4,1:COLOR 15,4,4:CLS
20 OPEN "GRP:" AS #1
30 ON SPRITE GOSUB 90:SPRITE ON
40 SPRITE$(0)=STRING$(8,255)
50 PUT SPRITE 0,(118,86),2,0
60 PUT SPRITE 1,(128,96),8,0
70 COLOR SPRITE(1)=&B01001000 ' <-----
80 GOTO 80
90 PSET (20,170),4
100 PRINT #1,"СТОЛКНОВЕНИЕ СПРАЙТОВ"
110 SPRITE OFF:RETURN
```

Если данный пример прогнать без строки 70, то на голубом фоне экрана появляются два пересекающихся спрайта: красный и зеленый. Поскольку в строке 30 назначено и включено прерывание по столкновению спрайтов, то с позиции (20, 170) будет выведено сообщение

СТОЛКНОВЕНИЕ СПРАЙТОВ

Пусть строка 70 в программе восстановлена. Тогда, учитывая инициализацию в 70 (бит 6 равен 1), выведенные пересекающиеся спрайты не вызовут обращения к подпрограмме обработки прерывания.

З а м е ч а н и е. Прерывание по столкновению спрайтов выполняется всегда правильно лишь при выключении компьютера из локальной вычислительной сети оператором __ NETEND.

Экранные страницы. Наличие видеопамяти объемом 128 Кбайт позволило организовать многостраничные экраны для режимов от SCREEN 5 до SCREEN 8 (см. табл. 24—26). Что представляют собой страницы? Фактически это отдельные непересекающиеся фрагменты видеопамяти. Для управления ими введен оператор

SET PAGE P1, P2 (3)

где: SET (установить), PAGE (страница) — служебные слова; P1, P2 — арифметические выражения.

При выполнении этого оператора используются целые части значений P1 и P2. Они должны лежать в диапазоне: для экранов 5—6 от 0 до 3, для экранов 7—8 от 0 до 1. По умолчанию P1=P2=0.

P1 называется *отображаемой страницей*. Ее содержимое индицируется в данный момент на экране.

P2 называется *активной страницей*. С ней проводится текущая работа по операторам программы, то есть вводятся тексты, формируются рисунки и т. п.

Отметим, что любой из параметров в операторе (3) действует до нового назначения по SET PAGE и смена страниц проводится довольно быстро.

В примере 12 показана работа с 4 страницами.

```
П Р И М Е Р  12

10 SCREEN 5:COLOR 15,4,4:CLS

20 DATA 1,3,8,15

30 FOR S=0 TO 3:SET PAGE 0,S:CLS:READC:R=S+1

40 LINE(30*R,18*R)-(60*R,48*R),C,BF

50 NEXT

60 FOR S=0 TO 3                          80 NEXT

70 SET PAGE S:BEEP:A$=INPUT$(1)          90 GOTO 60
```

По программе этого примера в строках 30—50 для режима SCREEN 5 формируется содержимое всех страниц от 0 до 3. Это квадраты разных размеров, положения и расцветки. Просмотр страниц реализуется по строкам 60—70. Для циклического перехода от одной страницы к другой надо нажать любую клавишу.

Имеются некоторые ограничения при совместном использовании спрайтов и экранных страниц. В этом случае вывод спрайта возможен только на странице 0, где он и должен быть сформирован.

В примере 13 демонстрируются спрайты на странице 0.

```
П Р И М Е Р  13

10 SCREEN 5,1:COLOR 15,4,4:CLS

20 SPRITE$(1)=STRING$(8,255):OPEN "GRP:"AS#1

30 SET PAGE 0,0:CLS

40 CIRCLE(120,100),50,15

50 PUT SPRITE 1,(112,92),8

60 PRESET(68,175):PRINT #1,"СТРАНИЦА (0,0)"

70 SET PAGE 0,3:CLS
```



```
80 CIRCLE(120,100),40,3,,,.5

90 LINE(70,75)-(170,125),7,B

100 PAINT(120,100),6,3

110 PRESET(68,175):PRINT #1,"СТРАНИЦА (3,3)"

120 SET PAGE 0:A$=INPUT$(1)

130 SET PAGE 3:A$=INPUT$(1)

140 GOTO 120
```

Обратите внимание, что при прогоне данной программы в соответствии с операторами строки 70 рисунок и подпись на третьей странице формируются при отображаемой нулевой странице. Поэтому упомянутые действия остаются как бы скрытыми от нас, и на экране дисплея мы их просто не замечаем. Затем созданное «невидимое» изображение проявляется по оператору SET PAGE 3 строки 130.

Копирование блоков. Прямоугольник на экранной странице со сторонами, параллельными осям координат, удобно задавать двумя точками: верхней левой (X1, Y1) и нижней правой (X2, Y2) вершинами. Будем называть такие прямоугольники блоками и обозначать их так: R=(X1, Y1) — (X2, Y2).

Широкий спектр для различного рода манипуляций с блоками предоставляет очень полезный и эффективный оператор COPY. С его помощью можно копировать экранные блоки на данной странице, перемещать их на другие страницы, упаковывать в цифровом виде в числовые массивы, записывать в виде файла данных на диск и осуществлять ряд других операций.

Записывается оператор COPY в одной из 7 следующих форм

1. COPY R, P1 TO (X, Y), P2, &
2. COPY R, P1 TO M
3. COPY R, P1 TO NAME
4. COPY M, F TO (X, Y), P2, &
5. COPY NAME, F TO (X, Y), P2, &
6. COPY M TO NAME
7. COPY NAME TO M

Здесь:

COPY (копировать), TO (в) — служебные слова;

X1, Y1, X2, Y2, P1, P2, X, Y, F — арифметические выражения, у которых используются целые части значений;

NAME — строковая переменная;

M — имя одномерного числового массива;

& — определитель цвета точек.

&{OR, AND, XOR, PSET, PRESET, TOR, TAND, TXOR, TPSET, TRESET).

Смысл этих параметров в случаях 1—7 таков:

R — блок;

P1, P2 — номера экранных страниц;

(X, Y) — точка на экране (странице);

F — указатель типа симметрии копируемого блока;

NAME — имя файла;

M — цифровой образ блока в виде значений элементов массива;

& — элемент для задания цвета копируемого блока в зависимости от его первоначальной раскраски и текущего цвета экрана.

Остановимся на описании семантики каждого из случаев 1—7.

1. При выполнении COPY содержимое блока R со страницы P1 пересылается на страницу P2 в блок с левой верхней вершиной (X, Y). По умолчанию P1 есть активная страница, а P2=P1. Параметр & определяет цвет CN каждой точки вновь создаваемого блока, исходя из цвета C этой точки в R и цвета S фона экрана. Делается это так, как указано в графе 3 таблицы 28, где NOT, OR и AND — обычные логические операции.

Определителю цвета может предшествовать префикс T. При копировании точек R, не являющихся «прозрачными», T никакого влияния не оказывает. Образы точек с прозрачной палитрой при наличии T раскрашиваются в цвет фона.

Таблица 28

Формирование цвета блока

Определитель цвета &		Получение цвета CN точек блока
1	PSET	C
2	PRESET	NOT C
3	OR	C OR S
4	AND	C AND S
5	XOR	(NOTC) AND S OR (NOTS) AND C

ПРИМЕР 14

```
10 SCREEN 5:COLOR 15,4,4:CLS
20 LINE(20,20)-(100,100),8,BF
30 COPY(60,70)-(140,120),0 TO (150,130),0,PRESET
40 GOTO 40
```


По данному фрагменту программы на нулевой странице экрана 5 рисуется квадрат красного цвета. Затем на эту же страницу копируется блок, имеющий непустое пересечение с квадратом. Образы точек блока оказываются раскрашенными в два цвета:

голубой; CN=NOT8=7;

светло-желтый; CN=NOT4=11.

В следующем примере демонстрируется копирование с одной страницы на другую.

ПРИМЕР 15

```
10 SCREEN 5:COLOR 15,4,4:CLS
20 SET PAGE 0,1:CLS
30 LINE(20,20)-(100,100),8,BF
40 SET PAGE 0,0:CLS
50 COPY(80,80)-(120,120),1 TO (150,130),0,PRESET
60 R$=INPUT$(1):SET PAGE 1
70 R$=INPUT$(1):SET PAGE 0:GOTO 60
```

2. При выполнении COPY содержимое R со страницы P1 копируется в числовой массив M. Оптимальный размер N массива, то есть наименьшая допустимая верхняя граница индексов его элементов, зависит от типа массива и режима экрана и определяется так, как это указано в таблице 29. Через T там обозначено количество точек копируемого блока R:

$$T=(ABS\ (X1-X2)+1)*(ABS\ (Y1-Y2)+1)$$

Таблица 29

Оптимальные размеры массива для копирований

Режим экрана		Целый	Размеры массива Вещественный
1	5, 7	(T+3)\4+1	(T+7)\16
2	6	(T+7)\8+1	(T+15)\32
3	8	(T+1)\2+1	(T+3)\8

3. При выполнении COPY содержимое блока R со страницы P1 копируется в виде файла данных под именем NAME на диск.

4. При выполнении COPY содержимое массива M помещается в некоторый блок страницы P2. Действие определителя цветов & здесь такое же, как и в случае 1. Положением блока и размещением информации в нем управляют параметры: точка

(X, Y) и указатель типа симметрии F. Остановимся на этом подробнее.

1). F=0. Блок в P2 формируется с левой верхней вершиной (X, Y) в том виде, как он был записан в M. Изображение на экране, получаемое от этого блока, обозначим через F0.

2). F=1. Местоположение и содержимое блока в P2 таково, что формируемое от него изображение получается из F0 осевой симметрией относительно прямой x=X.

3). F=2. Местоположение и содержимое блока в P2 таково, что формируемое от него изображение получается из F0 осевой симметрией относительно прямой y=Y.

4). F=3. Местоположение и содержимое блока в P2 таково, что формируемое от него изображение получается из F0 центральной симметрией относительно точки (X, Y).

Иллюстрацией к действию указателя симметрии F служит следующий пример.

```
П Р И М Е Р  16

10 SCREEN 5:COLOR 15,10,13:CLS:DEFINT A-Z
20 X1=60:Y1=60:X2=120:Y2=120:C=1
30 T=(ABS(X1-X2)+1)*(ABS(Y1-Y2)+1)
40 R=(T+3)\4+1:DIM L(R)
50 CIRCLE (X1,Y1),40,C:PAINT (X1,Y1),C
60 COPY(X1,Y1)-(X2,Y2),0 TO L
70 FOR F=0 TO 3
80 COPY L,F TO (150,130),0,XOR
90 R$=INPUT$(1):NEXT' НАЖИМАЙТЕ КЛАВИШУ !
100 GOTO 100
```

5. Выполнение COPY протекает точно так же, как и в предыдущем случае. Только данные здесь извлекаются не из массива, а с диска из файла "NAME".

6. При выполнении COPY числовой массив M записывается на диск под именем "NAME".

7. При выполнении COPY файл данных под именем NAME с диска записывается в массив M. Здесь необходимо согласовывать размеры M с объемом файла (см. пункт 2).

В примере 17 показаны различные типы копирований.


```
П Р И М Е Р  17

10 SCREEN 8:COLOR 15,130,1:CLS:DEFINT A-Z
20 X1=60:Y1=60:X2=120:Y2=120:C=13
30 T=(ABS(X1-X2)+1)*(ABS(Y1-Y2)+1)
40 R=(T+1)\2+1:DIM L(R),M(R)
50 CIRCLE (X1,Y1),40,C:PAINT (X1,Y1),C
60 COPY(X1,Y1)-(X2,Y2),0 TO L
70 FOR S=0 TO 3
80 COPY L,S TO (150,130),0,PRESET
90 BEEP:R$=INPUT$(1):NEXT
100 COPY L TO "TOR"
110 COPY "TOR" TO M
120 COPY M,0 TO (20,60),0,PRESET
130 GOTO 130
```

5.2. Виртуальный диск

Расширение объема ОЗУ в MSX-2 до 128 и более Кбайт не делает вновь появившееся свободное пространство памяти непосредственно доступным для интерпретатора Бейсика. По-прежнему для указания адреса отводится два байта, и потому прямая адресация возможна лишь в пределах от 0 до 65535. Как же использовать остальную память? Она рассматривается в качестве виртуального дискового устройства, называемого «диск ОЗУ», или, по-другому, "mem". В Бейсик добавлены 4 новых оператора расширения (__MEMINI, __MFILES, __MKILL, __MNAME). Остальные средства для работы с "mem" имеют синтаксис и семантику, в основном аналогичную соответствующим операторам и функциям в MSX DISK Бейсике. Однако слово "mem" перед именем файла здесь указывать обязательно. Например,

SAVE "mem: ВОЛНА"

Внимание! При выключении компьютера содержимое виртуального диска безвозвратно теряется.

Рассмотрим подробнее отмеченные выше операторы расширения.

CALL MEMINI. Данный оператор производит действия, аналогичные CALL FORMAT в MSX DISK Бейсике. По нему организуется форматирование устройства "mem". Выполнять CALL

MEMINI необходимо до первого обращения к "mem". Отметим, что формирование уничтожает все файлы, записанные на виртуальный диск ранее.

CALL MFILES. Выполнение этого оператора приводит к выводу на экран каталога файлов диска "mem". В отличие от FILES здесь операнды не допускаются. Нельзя использовать и префикс L для распечатки каталога на принтере.

CALL MKILL. При выполнении оператора CALL MKILL (NAME) с диска ОЗУ стирается файл под именем NAME, например:

```
CALL MKILL ("ТРОС")
__MKILL ("ДЫМ.2")
```

CALL MNAME. По оператору CALL MNAME ("имя 1" AS "имя 2") файл "имя 1" переименовывается в "имя 2".

Отметим два следующих обстоятельства.

1. Любые программы по оператору SAVE записываются на виртуальный диск в формате ASCII. Тем самым всегда возможно их слияние по MERGE.

2 Значения, возвращаемые функциями LOC, LOF и FPOS, могут вызвать переполнение при присваивании их целочисленным переменным. Поэтому для подобных целей лучше использовать переменные действительного типа.

5.3. Коды клавиш

Материал, излагаемый в этом пункте, относится к любой версии Бейсика.

Ранее мы встречались с двумя способами распознавания из программы кода ASCII вводимого с клавиатуры символа. Опираются они на использование функций INKEY$ и INPUT$( ).

ПРИМЕРЫ 18

```
A.    10 ' ФУНКЦИЯ  INKEY$. КУРСОРА НЕТ
      20 R$=INKEY$:IF R$="" THEN 20
      30 PRINT ASC(R$),R$:GOTO 20

B.    10 ' ФУНКЦИЯ  INPUT$. КУРСОР ЕСТЬ
      20 R$=INPUT$(1)
      30 PRINT ASC(R$),R$:GOTO 20
```

Разница в этих вариантах не только в наличии или в отсутствии на экране курсора при реализации строки 20, но и в том, что в случае А, ожидая нажатия клавиши, процессор продолжает выполнять программу, а в случае В он находится в режиме ввода символа. С одной стороны, оба этих способа идентификации клавиш и вводимых по ним символов просты и удобны. Однако по меньшей мере два факта заставляют нас применять для подобных целей и иные приемы. Рассмотрим их.

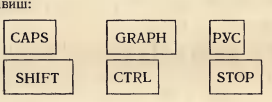
1. Некоторые коды ASCII в диапазоне от 0 до 31 вырабатываются лишь при одновременном нажатии клавиши CTRL и определенного символа. Часть этих кодов является управляющими сигналами, а остальные можно использовать при написании программ (см. табл. 30).

Таблица 30

Коды для двух клавиш

Клавиша		Символ	Код	Примечание	Клавиша		Символ	Код	Примечание
1	CTRL	D	04		7	CTRL	V	22	
2	CTRL	O	15		8	CTRL	W	23	
3	CTRL	P	16		9	CTRL	X	24	=SELECT
4	CTRL	Q	17		10	CTRL	Y	25	
5	CTRL	S	19		11	CTRL	Z	26	
6	CTRL	T	20		12	CTRL	C	27	=ESC

2. Шесть клавиш:



вообще не имеют кодов ASCII, и потому они «не читаемы» функциями INKEY$ и INPUT$ ( ). Однако распознать их из программы все же можно. Дело в том, что информация о последней нажатой клавише (именно клавише, а не символе!) хранится в буфере NEWKEY, расположенном с адреса &HFBE5.

Для клавиатуры без выделенной цифровой зоны NEWKEY занимает 9 последовательных байт памяти:

&HFBE5 — &HFBED

Для клавиатуры с дополнительной правой цифровой зоной буфер NEWKEY на два байта длиннее и расположен по адресам:

&HFBE5 — &HFBEF (7)

Каждой клавише соответствует ровно один бит NEWKEY, и при ее нажатии значение этого бита сбрасывается на нуль. При этом биты буфера для ненажатых клавиш становятся равными 1. Если (7) представить в виде матрицы с 11 строками (байты) и 8 столбца

ми (биты), то положение нулей (X, Y) ($0 \leqslant X \leqslant 7$, $0 \leqslant Y \leqslant 11$) в ней и будет определять совокупность всех нажатых в данный момент клавиш. В таблице 31 указаны клавиши, дающие нулевое значение своим битам.

Запустив на счет фрагмент программы примера 19 и нажимая те или иные клавиши, можно проследить за изменением матрицы клавиатуры и тем самым проверить таблицу 31.

ПРИМЕРЫ 19

```
10 SCREEN 0:COLOR 15,4,4:WIDTH 80
20 A=&HFBE5:A$="00000000"
30 LOCATE 8,3:PRINT "!  МАТРИЦА   ! Y ! X"
40 LOCATE 6,4:GOSUB 140:LOCATE 6,16:GOSUB 140
50 FOR Y=0 TO 10
60 Z=PEEK(A+Y):Z$=RIGHT$(A$+BIN$(Z),8)
70 LOCATE 8,Y+5:PRINT "!  "+Z$+"  !      !";
80 LOCATE 28,Y+5:PRINT SPACE$(16);
85 IF Z=255 THEN 130
90 BEEP:LOCATE 22,Y+5:PRINT Y:LOCATE 28,Y+5
100 FOR X=1 TO 8
110 IF MID$(Z$,X,1)="0" THEN PRINT 8-X;
120 NEXT X
130 NEXT Y:GOTO 50
140 PRINT STRING$(30,"-"):RETURN
```

По этой программе сканируется буфер NEWKEY и на экране индицируется как его текущее состояние, так и координаты (X, Y) нулевых битов. В примере 20 показаны переходы по клавишам

SHIFT, CTRL, GRAPH, CAPS и РУС.

ПРИМЕР 20

```
10 SCREEN 0:COLOR 15,4,4:KEY OFF
20 E=PEEK(&HFBEB)
30 IF E=254 THEN 100' ----> "SHIFT"
40 IF E=253 THEN 200' ----> "CTRL"
```


Таблица 31

Матрица клавиатуры

Адрес (байт)	X (биты) 7	6	5	4	3	2	1	0	Y
FBE5	6	5	4	3	2	1	+	9	0
FBE6	V	*	H	|	=	0	8	7	1
FBE7	I	F	?	—	@	B	>	\	2
FBE8	0	[	R	\/	A	U	W	S	3
FBE9	K	J	Z	;	T	X	D	L	4
FBEA	Q	N	—	C	M	G	E	Y	5
FBEB	F3	F2	F1	РУС	CAPS	GRAPH	CTRL	SHIFT	6
FBEC	RETURN	SELECT	BS	STOP	TAB	ESC	F5	F4	7
FBED	→	↓	↑	←	DEL	INS	HOME	ПРОБЕЛ	8
FBEE	4	3	2	1	0	*	+	RETURN	9
FBEF	.	,	—	9	8	7	6	5	10

```
50 IF E=251 THEN 300' ---> "GRAPH"
60 IF E=247 THEN 400' ---> "CAPS"
70 IF E=239 THEN 500' ---> "РУС"
80 GOTO 20
```

5.4. Инициализация компьютера

Включение компьютера автоматически приводит его в некоторое исходное состояние: назначается режим экрана, устанавливается его ширина, формируются цвета фона и изображения, фиксируется значение подсказки и определяются многие другие параметры. В MSX-2 имеется ряд дополнительных операторов, проводящих начальную инициализацию, которая не разрушается при включениях-выключениях компьютера. Устанавливаемые ими назначения сохраняются в специальном небольшом участке памяти за счет питания от сменной аккумуляторной батарейки. Речь идет о 10 операторах, к описанию которых мы и приступим.

Подсказка, заголовок, пароль. Приглашение-подсказку "Ok" можно заменить последовательностью любых других символов, длиной не более 6. По оператору SET PROMPT & в качестве подсказки фиксируется значение строкового выражения & или первые 6 символов этого значения, если его длина больше 6 (*set* — установить, *prompt* — подсказка).

По оператору SET TITLE &, C переопределяется «заголовок» титульного экрана-заставки, появляющийся при включении компьютера.

Здесь:

title (заголовок) — служебное слово;

& — строковое выражение, у которого для заголовка используется до 6 первых символов значения;

C — арифметическое выражение со значениями от 1 до 4.

Заголовок располагается на экране после всех сообщений по центру фона. Если его длина равна 6, то для продолжения работы требуется «подтолкнуть» систему, нажав на любую клавишу. Параметр C не является обязательным. По умолчанию он равен 1 и служит для фиксирования раскраски экрана-заставки по одной из 4 возможных схем. По оператору

SET PASSWORD &

можно задать пароль, являющийся значением строкового выражения & (*password* — пароль). Длина этого значения не должна быть более 255.

Пусть пароль зафиксирован. Тогда каждый раз при включении компьютер будет требовать от пользователя его ввода с клавиатуры. А что делать, если пароль забыт? Необходимо осуществить его сброс. Для этого достаточно выполнить такую последовательность действий.

а). При включении компьютера держать одновременно нажатыми клавиши STOP и GRAPH. Это не отменит пароля, но приведет к обходу экрана-заставки.

б). Выполнить оператор SET PROMPT "Ok". Теперь уже пароль будет отменен. Дело все в том, что из трех рассмотренных в этом пункте установок — подсказки, заголовка и пароля — действует только одна, назначенная последней.

Звук, центрирование фона. Тон звука, выводимого по оператору BEEP, и его громкость можно переопределять оператором

SET BEEP звук, громкость

Здесь «звук» и «громкость» — арифметические выражения, у которых используются целые части значений. Лежать они должны в диапазоне от 0 до 3. Один из этих параметров может быть опущен.

Если фон, на котором выводится изображение, плохо отцентрирован на экране, то поправить это можно оператором

SET ADJUST (X, Y)

Здесь:

adjust (регулировать) — служебное слово;

X, Y — арифметические выражения со значениями от — 7 до 8.

Параметры экрана. Многие текущие параметры, связанные с экраном, можно сохранить для начальной инициализации компьютера. Для этого достаточно выполнить оператор SET SCREEN. Сразу же оговоримся, что запоминаются только те назначения, которые фиксируются операторами:

SCREEN, COLOR, WIDTH и KEY ON/OFF

выполненными в непосредственном счете. Речь идет о таких параметрах:

M — режим экрана (0, 1);

S — размер и масштаб спрайтов;

K — наличие (1) или отсутствие (0) отзвука клавиш;

V — скорость обмена с кассетным накопителем (1 — 1200 бод, 2 — 2400 бод);

P — варианты принтера (0 — MSX, +0 — другое печатающее устройство);

R — режим вывода на монитор (0—3; 0 — нормальный);
C1, C2, C3 — цвета изображения фона и бордюра (0—15);
ширина экрана — 1—80 при M=0, 1—40 при M=1;
строка-подсказка — есть или нет.

Напомним, что первые 6 параметров можно задавать оператором SCREEN:

```
SCREEN M, S, K, V, P, R
```

Цвета изображения, фона и бордюра формируются оператором COLOR C1, C2, C3. Ширина α экрана назначается оператором WIDTH α. Отключение и подключение строки-подсказки на текстовых экранах 0 и 1 реализуется соответственно операторами KEY OFF и KEY ON.

Дата, время. Дата устанавливается оператором

```
SET DATE "ММ/ДД/ГГ"          (8)
```

Здесь "ММ/ДД/ГГ" — формат (месяц, день, год). При записи SET DATE вместо М, Д, Г должны быть проставлены десятичные цифры. Нули указывать обязательно. Если оператор (8) выполнен, то фактически запущен автоматический календарь, работающий автономно вне зависимости от включений и выключений компьютера.

По оператору GET DATE t (*get* — получить) строковая переменная t принимает из календаря текущее значение даты в том формате, в каком она фиксировалась в (8).

Время задается оператором

```
SET TIME "ЧЧ:ММ:СС"          (9)
```

Здесь: "ЧЧ:ММ:СС" — формат (часы, минуты, секунды). При записи SET TIME вместо Ч, М, С должны быть проставлены десятичные цифры. Нули указывать обязательно. Если оператор (9) выполнен, то фактически запущены электронные часы, работающие вне зависимости от включений и выключений компьютера.

По оператору GET TIME t строковая переменная t получает текущее значение времени в том формате, в каком оно фиксировалось в (9).

Отметим, что форматы в (8) и (9) можно задавать не только константами, но и произвольными строковыми выражениями.

Остановимся еще на одном варианте рассмотренных в этом пункте операторов. Если наряду с (8) и (9) выполнить еще два оператора:

```
SET TIME время, A
SET DATE дата, A             (10)
```


то зафиксированные в них новые «дата» и «время» будут рассматриваться как дата и время подачи сигнала. Таким образом мы как бы имеем в наличии «будильник с календарем». Правда, автоматически такой будильник работать не будет. Необходимо, например, в подпрограмме организовать сравнение текущих даты и времени с соответствующими параметрами, заданными в (10), и при их совпадении выдавать серию сигналов BEEP, проигрывать какой-либо музыкальный фрагмент или выводить некоторое сообщение.

При использовании параметра A всегда следует вводить сначала время, а затем дату.

Возвратить параметры «время» и «дату», сформированные по (10), можно по операторам

```
GET TIME &1, A
GET DATE &2, A               (11)
```

Правда, здесь необходимо учитывать некоторые нюансы. Во-первых, строковой переменной &1 присваивается значение без учета секунд ("ЧЧ:ММ—ØØ:"). Во-вторых, переменная &2 получает значение без учета месяца и года ("ØØ/ДД/ØØ").

```
П Р И М Е Р  21
10 SCREEN 0:COLOR,15,4,4:KEY OFF
20 SET DATE "03/08/90":SET TIME "13:44:33"
30 '------------------------------------------
40 SET TIME "13:45:33",A: GET TIME B$,A
50 GOSUB 70: GOTO 50
60 '------------------------------------------
70 GET DATE D$: GET TIME T$
80 LOCATE 15,18: PRINT D$,,T$
90 IF T$=B$ANDMID$(D$,4,2)="08" THEN GOSUB110
100 RETURN
110 FOR S=1 TO 25:BEEP:NEXT:RETURN
```

5.5. Локальная вычислительная сеть

Компьютеры "Ямаха" (MSX-2) объединяются в локальную вычислительную сеть, в которой одновременно может присутствовать один «учительский» и до 15 «студенческих» компьютеров. Сеть поддерживается MSX-DOS, MSX-CP/M и Бейсик-систе-

мой. В дальнейшем нас будет интересовать лишь последний случай. Сразу же оговоримся, что рассматриваемая сеть существенно отличается от сети версии 1 (MSX-1). Здесь гораздо больше команд, иной синтаксис и значительно расширены предоставляемые пользователю возможности.

Везде ниже учительский компьютер будем обозначать через *T*, а студенческие — через *P*. Для идентификации за *T* закрепляется номер 0, а за *P* — номера 1, 2, ..., 15. Номер каждого компьютера индицируется на экране дисплея при запуске Бейсик-системы. Впрочем, в любой момент его можно узнать и по запросу __WHO[1]. Кроме того, по команде __WHO (V) номер компьютера присваивается числовой переменной V.

Список команд. В таблице 32 приведен полный список из 25 сетевых команд и знаком "+" отмечено, какие из них доступны *T* и *P*. При описании форматов команд использованы такие обозначения:

name	— имя файла;
n	— номер студенческого компьютера;
m	— номер строки программы;
A, A1, A2	— адреса оперативной памяти, видеопамяти или сетевого ОЗУ;
V, V1, V2	— числовые переменные;
W	— арифметическое выражение;
mes	— пересылаемое сообщение;
command	— команда;
S	— опция, указывающая на видеопамять (S — screen);
N	— опция, указывающая на работу с сетевым ОЗУ (N — net).

Здесь:

n, m, A, A1, A2, W	— произвольные арифметические выражения;
V, V1, V2	— числовые переменные;
name, mes, command	— строковые выражения.

Мы не будем подробно останавливаться на описании синтаксиса и семантики каждой сетевой команды, а ограничимся рассмотрением конкретных примеров. Заметим, что ключевые слова команд записывать в тексте программы и набирать на клавиатуре можно в сокращенном виде. После символа "__" (CALL) достаточно указывать лишь первые 4 буквы слова. Например, __MESS (7).

Каждый студенческий компьютер может находиться в двух состояниях (режимах). В одном из них для *P* доступен лишь минимальный набор команд MIN, а в другом — расширенный набор команд MAX. Эти случаи отмечены соответственно в колонках 4 и 5 таблицы 32. Запуск Бейсик-системы на *P* устанавливает его в состояние с минимальным набором сетевых команд MIN. Смена режимов осуществляется из компьютера *T* по командам __ENACOM и __DISCOM

[1] Символ подчеркивания "—" заменяет слово CALL.

Пример 1. Все нижеприведенные команды подаются из *T*. Реализуются они или в непосредственном счете, или в программном режиме.

А. Перевод компьютеров *P* в состояние с расширенным набором команд MAX.

команда	комп'ютер
_ENACOM(1Ø)	- 1Ø
_ENACOM(Ø)	- все студенческие
_ENAC(Ø)	- все студенческие
X=8:_ENAC(X)	- 8

В. Перевод компьютеров *P* в состояние с минимальным набором команд MIN.

команда	комп'ютер
_DISCOM(9)	- 9
_DISCOM(Ø)	- все студенческие
_DISC(Ø)	- все студенческие
Y=2:_DISC(Y)	- 2

Список команд, доступных в текущий момент на данном компьютере, можно вывести на его текстовые экраны 0 или 1 по команде __HELP (*help* — помощь).

Поэкспериментируйте с этой командой на *P* и *T*, выполняя на *T* поочередно команды __ENACOM и __DISCOM.

Инициализация и состояние сети. При включении компьютера *T* проводится автоматическая инициализация сети, то есть по умолчанию выполняется команда

__NETINIT (1)

(*net* — сеть, *initiation* — инициализация).

Не все прикладные программы могут работать в сетевом режиме. Для «разрушения» сети и перевода компьютеров в индивидуальный режим достаточно выполнить на *T* команду __NETEND (*net* — сеть, *end* — конец).

Таблица 32

Список сетевых команд

	Команда		T	P (MIN)	P (MAX)
1	__HELP		+	+	+
2	__WHO	(V)	+	+	+
3	__ENACOM	(n)	+	–	–
4	__DISCOM	(n)	+	–	–
5	__NETINIT		+	+	+
6	__NETEND		+	+	+
7	__ONLINE		–	+	+
8	__OFFLINE		–	+	+
9	__PON		+	–	–
10	__POFF		+	–	–
11	__CHECK	(V1, V2)	+	–	–
12	__SEND	(name, n)	+	–	+
13	__SNDRUN	(name, n)	+	–	–
14	__BSEND	(name, n, A1, A2, S)	+	–	+
15	__RECEIVE	(name, n)	+	–	+
16	__BRECEIVE	(name, n, A1, A2, N)	+	–	+
17	__SNDCMD	(command, n)	+	–	–
18	__RUN	(n, A, m)	+	–	–
19	__STOP	(n)	+	–	–
20	__SNDMAIL	(n)	+	–	–
21	__RCVMAIL	(n)	+	–	–
22	__MESSAGE	(mes, n)	+	–	–
23	__TALK	(mes, V)	–	+	+
24	__PEEK	(V, A, n, N)	+	+	+
25	__POKE	(w, A, n, N)	+	+	+

Например, для запуска на *T* встроенной программы PAINTER (художник) требуется реализовать последовательность команд:

__NETEND: __PAINTER

Последующий возврат в сетевой режим осуществляется по команде (1).

Замечание. Выполнение команды __NETEND на конкретном компьютере приводит к увеличению скорости вычислений на нем до 25%. Это следует учитывать при длительной работе в несетевом режиме.

При включении любого компьютера *P* проводится автоматическое подключение его к сети, то есть по умолчанию выполняется команда

__ONLINE (2)

Для отключения *P* от локальной сети и перевода его на индивидуальный режим работы требуется выполнить на *P* команду __OFFLINE. При этом прекращается всякий обмен в сети с данным компьютером, и тем самым он становится защищенным от различного рода вмешательств в работу и прерываний со стороны *T* или других *P*. Скорость вычислений на *P* не изменяется. Последующее включение *P* в сеть осуществляется по команде (2).

Перед началом работы *T* с *P* или *P* с *P* на *T* необходимо «активировать» опрос, то есть выполнить команду

__PON (3)

Отменяется действие (3) по команде __POFF. Отметим, что если команда (3) не реализована, то не гарантируется возможность правильного определения из *T* номеров, подключенных к сети компьютеров *P*.

Следующая очень важная команда __CHECK (*check* — проверка) позволяет нам выяснить на *T*, какие из компьютеров *P* подключены к сети и разрешен ли им обмен с другими *P* (__ENACOM, __DISCOM). Записывается команда в одной из трех форм:

__CHECK (V1, V2) (4)

__CHECK (V1) (5)

__CHECK (, V2) (6)

где V1 и V2 — числовые переменные.

При выполнении (4) и (5) в двоичном представлении значения V1 отдельными битами фиксируется подключение или неподключение *P* к локальной сети. При этом, если бит *n* (n=1,2...,15) начиная от младшего (правого) бита равен нулю, то компьютер номер n подключен к сети. В противном случае он или вообще выключен, или находится в индивидуальном режиме (__OFFLINE)

Пример 2. При запуске нижеприведенного фрагмента программы мы видим, что к сети подключены компьютеры 1 и 3-10.

```
10 _CHECK(A)                    run
20 PRINT BIN$(A)          1111100000000010
```

При выполнении (4) и (6) в двоичном представлении значения V2 отдельными битами фиксируется разрешение на минимальный (__DISCOM) или расширенный (__ENACOM) набор сетевых команд. При этом если бит n (n=1,2, ..., 15) начиная от младшего (правого) - бита равен нулю, то компьютеру номер n разрешен расширенный набор команд. В противном случае для него доступен лишь минимальный набор команд.

Пример 3. При выполнении фрагмента

мы видим, что расширенный набор команд доступен компьютерам 1—3 и 13—15.

```
10 _CHECK(,B)
20 B$=RIGHT$("0000000000000000"+BIN$(B),15)
30 PRINT B$
run
000111111111000
```

Пример 4. Составить программу для *T*, по которой каждому подключенному к сети компьютеру *P* с четным номером будет разрешен расширенный набор команд, а состояние других компьютеров не изменится.

Решение данной задачи получается, например, по такому фрагменту:

```
10 _CHECK(D)
20 D$=RIGHT$(STRING$(15,"0")+BIN$(D),15)
30 FOR K=2 TO 15 STEP 2
40 IF MID$(D$,16-K,1)="0" THEN _ENACOM(K)
50 NEXT
```

Передачи программ и данных из Т. С учительского компьютера *T* программу на Бейсике можно пересылать в *P* прямо с диска или из оперативной памяти. Делается это по команде __SEND (*send* — посылать), записываемой в одной из следующих 4 основных форм:

__SEND (name, n) (7)

__SEND (name, 0) (8)

__SEND (, n) (9)

__SEND (, 0) (10)

Рассмотрим семантику каждой из команд (7) — (10).

По команде (7) программа name с диска пересылается в компьютер номер n (n=1, 2, ..., 15).

По команде (8) программа name с диска пересылается во все студенческие компьютеры *P*.

По команде (9) программа из оперативной памяти *T* пересылается в компьютер номер n (n=1, 2, ..., 15).

По команде (10) программа из оперативной памяти *T* пересылается во все студенческие компьютеры *P*.

При любых пересылках содержимое ОЗУ в *T* не меняется. Во время передачи на экране *P* высвечивается предупреждающее сообщение WAIT (ждите). Прежняя программа и данные в ОЗУ *P* теряются.


```
A.  _SEND("СЛИВА",7)
```

```
B.  A$="ВИШНЯ":_SEND(A$,0)
```

С. В этом примере с диска в *P* с номерами от 1 до 10 пересылаются соответственно программы:

ЗАДАЧА. 1, ЗАДАЧА. 2, ..., ЗАДАЧА. 10

```
10 X$="ЗАДАЧА."
20 FOR S=1 TO 10
30 A$=MID$(STR$(S),2)
40 _SEND(X$+A$,S)
50 NEXT
```

Если пересылаемая из *T* в *P* программа должна быть сразу же автоматически запущена на счет, то вместо соответствующей команды __SEND нужно использовать команду __SNDRUN (*send* — посылать, *run* — запуск). По аналогии с (7) — (10) для __SNDRUN имеются следующие формы записи:

__SNDRUN (name, n)
__SNDRUN (name, 0)
__SNDRUN (, n)
__SNDRUN (, 0)

Передача программ, данных или содержимого видеопамяти в машинном коде из *T* в *P* реализуется по команде

__BSEND (name, n, A1, A2, S) (11)

Все параметры в (11) не являются обязательными. Если n отсутствует или на его месте стоит нуль, то пересылка осуществляется во все студенческие компьютеры.

Рассмотрим некоторые конкретные случаи.

Случай 1.

__BSEND (name, n, A1) (12)

При выполнении (12) программа name в машинных кодах или данные, записанные из Бейсика по BSAVE, __BSEND или __BRECEIVE, пересылаются с диска в оперативную память студенческого компьютера n начиная с адреса A1.

Пример 6.

```
_BSEND("SHIP.BIN",3,&H8000)
```

Случай 2.

__BSEND (, n, A1, A2, S) (13)

При выполнении (13) содержимое видеопамяти (экрана) компьютера *T* в диапазоне адресов от A1 до A2 передается по этим же адресам в видеопамять студенческого компьютера п. Перед передачей в принимающем компьютере автоматически устанавливается требуемый режим видеопроцессора (экрана). По умолчанию

A1 =& H0000, A2 =& HFFFF

однако целесообразней A1 и A2 задавать в явном виде с учетом режима экрана, его ширины и страницы (см. табл. 33).

Таблица 33

Адреса видеопамяти

Экран	Ширина	Номер страницы	Адреса видеопамяти	
			Начало	Конец
1 0	40	—	0	& HFFF
2 0	80	—	0	& H1FFF
3 1	—	—	0	& H3BFF
4 2, 3, 4	—	—	0	& H3FFF
5 5, 6	—	1	0	& H7FFF
6 5, 6	—	2	& H8000	& HFFFF
7 5, 6	—	3	& H10000	& H17FFF
8 5, 6	—	4	& H18000	& H1FFFF
9 7, 8	—	1	0	& HFA9F
10 7, 8	—	2	& H10000	& H1FFFF

Пример 7

```
A.      _BSEND(,,&H0000,&h1FFF,S)
B.      _BSEND(,9,,,S)
```

С. В этом примере на графическом экране *T* формируется рисунок и затем изображение передается в компьютер 10.

```
10 ' Убегающий квадрат
20 SCREEN 5:COLOR 15,4,4
30 DEFINT A-I,L-Z
40 X=48   'абсцисса левого угла
50 N=25   'число квадратов
60 K=.9   'коэффициент деления стороны
70 L=256-2*X:Y=(192-L)\2
```



```
80 ' Вершины исходного квадрата
90 X(0)=X:X(1)=X+L:X(2)=X+L:X(3)=X
100 Y(0)=Y:Y(1)=Y:Y(2)=Y+L:Y(3)=Y+L
110 FOR U=1 TO N
120 ' Рисование квадрата
130 FOR S=0 TO 3:T=(S+1)MOD4
140 LINE (X(S),Y(S))-(X(T),Y(T))
150 NEXT
160 ' Вершины очередного квадрата
170 A=X(0):B=Y(0)
180 FOR S=0 TO 2
190 X(S)=X(S)+K*(X(S+1)-X(S))
200 Y(S)=Y(S)+K*(Y(S+1)-Y(S))
210 NEXT
220 X(3)=X(3)+K*(A-X(3))
230 Y(3)=Y(3)+K*(B-Y(3))
240 NEXT U
250 ' Пересылка рисунка
260 _BSEND (,10,0,&H7FFF,S)
270 GOTO 270
```

Случай 3 (см. 11).

Здесь файл паπе должен быть сформирован командой BSAVE или сетевыми командами __BSEND или __BREC с параметром S. Выполнение (11) приводит к передаче файла паπе с диска в видеопамять компьютеру п в диапазоне адресов от A1 до A2. При пересылке требуемый режим экрана автоматически не назначается. Иными словами, его необходимо установить на п до выполнения __BSEND. По умолчанию A1 принимается равным начальному адресу, с которого данные записывались на диск. Если этот адрес окажется больше A2, то A2 игнорируется.

ПРИМЕРЫ 8

```
A.      _BSEND("V.BIN",11,,,S)
B.      _BSEND("DD",,,&H0000,&H17FFF,S)
```

Передача программ и данных из *P*. Пусть конкретный студенческий компьютер q (q=1, 2, ..., 15) находится в состоянии __ENACOM. Тогда для q доступны сетевые команды __SEND и __BSEND и с него можно организовать пересылку программ и данных из оперативной памяти и видеопамяти в другие компьютеры *P* и *T*.

Команда __SEND записывается в одной из трех форм:

__SEND (, n)
__SEND (, 0)
__SEND (name, 0)

В любом из этих вариантов при выполнении __SEND пересылается программа из ОЗУ компьютера q.

В первом случае пересылка проводится в студенческий компьютер n (n≠0, n≠q).

Во втором случае пересылка организуется в ОЗУ *T*.

В третьем случае пересылаемая программа записывается на диск под именем name. Оперативная память *T* не изменяется.

Команда __BSEND также записывается в трех возможных формах:

__BSEND (, n, A1, A2, S)
__BSEND (, 0, A1, A2, S)
__BSEND (name, 0, A1, A2, S)

В любом из этих вариантов при выполнении __BSEND пересылается содержимое видеопамяти компьютера q в диапазоне адресов от A1 до A2.

В первом случае пересылка организуется в видеопамять студенческого компьютера n (n≠0, n≠q) с автоматическим настроем в n требуемого режима экрана.

Во втором случае пересылка организуется в видеопамять *T* с автоматическим настроем там требуемого режима экрана.

В третьем случае пересылаемые данные записываются на диск под именем name. Режим экрана не запоминается. Оперативная память в *T* не изменяется.

ПРИМЕР 9

```
A.  _SEND(,7)          C.  _BSEN(,7,&H8000,&HFFFF,S)
B.  _SEND(,0)          D.  _BSEN(,0,&H0000,&HFA9F,S)
```

Прием программ и данных в *T*. С учительского компьютера *T* программу на Бейсике можно принять от любого студенческого компьютера *P*, подключенного к сети. Делается это по команде __RECEIVE (*receive* — получать, принимать), записываемой в одной из следующих двух форм:

__RECEIVE (name, n) (14)

__RECEIVE (, n) (15)

Здесь name и n имеют тот же смысл, что и в команде __SEND.

Рассмотрим семантику каждой из этих команд.

По команде (14) программа из ОЗУ студенческого компьютера *P* с номером n (n=1, 2, ..., 15) записывается на диск под именем name. Содержимое ОЗУ компьютера *T* не меняется.

По команде (15) программа из ОЗУ студенческого компьютера *P* с номером n (n=1, 2, ..., 15) пересылается в ОЗУ компьютера *T*. Прежняя программа и данные в *T* теряются.

При пересылках из *P* в *T* работающая в *P* программа прерывается, ОЗУ для *P* не изменяется и во всех случаях на экране *P* высвечивается сообщение WAIT (ждите).

ПРИМЕР 10

```
A.  _RECE("ИГРА",9)    B.  _RECE(,9)
```

Здесь в первом случае программа из компьютера 9 записывается на диск под именем «ИГРА», а во втором случае эта же программа пересылается в оперативную память *T*.

Прием программ, данных или содержимого видеопамяти из *P* в *T* в машинном коде реализуется командой

__BRECEIVE (name, n, A1, A2, S) (16)

Здесь все параметры в __BREC, кроме n, являются необязательными, а их смысл тот же самый, что и в команде __BSEND.

Рассмотрим некоторые случаи.

Случай 1.

__BREC (name, n, A1, A2) (17)

При выполнении (17) программа или данные в машинных кодах из диапазона адресов от A1 до A2 ОЗУ студенческого компьютера с номером n пересылаются в *T* и записываются на диск под именем name. Содержимое оперативной памяти передающего и принимающего компьютеров не изменяется. Если записывалась программа, то в последующем ее стартовым адресом для команды BLOAD будет A1.

ПРИМЕР 11

```
_BREC("ЦИРК",3,&H8000,&H9000)
```

Случай 2.

__BREC (, n, A1, A2, S) (18)

При выполнении (18) содержимое видеопамяти (экрана) студенческого компьютера с номером n в диапазоне адресов от A1 до A2 пересылается в видеопамять *T* по тем же самым адресам. Предварительно в *T* автоматически устанавливается требуемый режим видеопроцессора.

ПРИМЕР 12

```
_BREC(,4,,,S)          C.    _RUN
```

Случай 3 (см. (16)).

При выполнении (16) содержимое видеопамяти студенческого компьютера с номером n в диапазоне адресов от A1 до A2 принимается и записывается в машинных кодах на диск под именем name. В дальнейшем при использовании name командой BLOAD требуется предварительная установка необходимого режима экрана.

Прием программ и данных в *P*. Пусть студенческий компьютер q (q=1, 2, ..., 15) находится в состоянии __ENACOM, то есть ему доступен расширенный набор сетевых команд. Тогда, в частности, можно использовать и команды __RECEIVE и __BRECEIVE для приема в q программ и данных из оперативной памяти и видеопамяти других компьютеров *P* и *T*.

Команда __RECEIVE записывается в одной из возможных форм:

__RECE (, n)
__RECE (, 0)
__RECE (name, 0)

В любом варианте при выполнении __RECE в студенческий компьютер q принимается некоторая программа.

В первом случае прием проводится из компьютера n (n≠0, n≠q).

Во втором случае прием организуется из ОЗУ *T*.

В третьем случае принимается программа, записанная на диске под именем name. Оперативная память *T* не изменяется.

Команда __BRECEIVE записывается в следующих формах:

__BREC (n, A1, A2, S)
__BREC (, 0, A1, A2, S)
__BREC (name, 0, A1, A2, S)

В любом из этих вариантов при выполнении __BREC в видеопамяти студенческого компьютера q в диапазоне адресов от A1 до A2 принимается набор данных.

В первом случае прием организуется из видеопамяти компьютера n (n≠0, n≠q) из диапазона адресов от A1 до A2 с автоматическим настроем в q требуемого режима экрана.

Во втором случае прием проводится из видеопамяти *T* из диапазона адресов от A1 до A2 с автоматическим настроем в q требуемого режима экрана.

В третьем случае данные принимаются с диска, записанные там командами __BSEND или __BREC с параметром S под именем name. Режим экрана в q требует предварительной установки. Оперативная память в *T* не изменяется.

Пример 13.

```
A.    _BREC(,7,,,S)
B.    _BREC(,0,,,S)
```

C. Пусть на компьютере 15 запущена на счет программа

```
C.    10 SCREEN 0:COLOR 15,4,4:WIDTH 40
      20 PRINT "... искать же чего-либо -"
      30 PRINT "    хотя бы грибов или  "
      40 PRINT "    какую-либо завимость "
      50 PRINT "    нельзя иначе, как   "
      60 PRINT "    смотря и пробуя.    "
```

Если на компьютере 5 выполнить команду

__BREC (, 15, 0, &H777, S)

то на его экране появится высказывание Д. И. Менделеева, набранное в операторах PRINT строк 20—60 и лучше всего объясняющее читателю, как усвоить сетевые команды.

Работа с байтами данных. Среди сетевых средств имеются и такие, которые позволяют непосредственно обращаться к байтам данных оперативной памяти (&H8000—&HFFFF) или сетевого ОЗУ (&H7800—&H7FFF) своего компьютера. Кроме того, из *T* подобное обращение возможно и к студенческим компьютерам *P*. Речь идет о командах __POKE и __PEEK.

Команда __POKE имеет такие формы записи:

__POKE (w, A) (для *T* и *P*) (19)

__POKE (w, A, n) (для *T*) (20)

__POKE (w, A, n, N) (для *T*) (21)

Остановимся на каждой из них в отдельности.

Команда (19) используется в любом компьютере для записи значения арифметического выражения w в байт памяти по адресу A своего сетевого ОЗУ.

Команда (20) используется в компьютере *T* для записи значения w в байт оперативной памяти по адресу A студенческого компьютера с номером n.

Команда (21) используется в компьютере *T* для записи значения w в байт сетевого ОЗУ по адресу A студенческого компьютера с номером n.

ПРИМЕР 14

```
A.        _POKE(222,&H7900)
B.     W=50:_POKE(W,&HBBBB,5)
C.     W=17:_POKE(W+2,&H7800,3,N)
```

Команда __PEEK (v, A) имеет следующие формы записи

```
__PEEK (v, A)          (для T и P)      (22)
__PEEK (v, A, n)       (для T)          (23)
__PEEK (v, A, n, N)    (для T)          (24)
```

Опишем семантику этой функции для каждого из отмеченных случаев в отдельности.

Команда (22) используется в любом компьютере. При ее выполнении переменная v получает значение, равное числу из байта своего сетевого ОЗУ по адресу A.

Команда (23) используется в компьютере *T*. При ее выполнении переменная v получает значение, равное числу из байта оперативной памяти по адресу A студенческого компьютера с номером n.

Команда (24) используется в компьютере *T*. При ее выполнении переменная v получает значение, равное числу из байта сетевого ОЗУ по адресу A студенческого компьютера с номером n.

ПРИМЕР 15

```
A.      _PEEK(X,&H7800):PRINT X
B.      _PEEK(Y,&H8000,7)
C.      _PEEK(Z,&H7FFF,8,N)
```

Более содержательные примеры с командами __POKE и __PEEK будут приведены в последующих пунктах.

Передача команд из *T*. Любую команду на Бейсике можно переслать из учительского компьютера *T* для выполнения в *P*. Делается это по команде.

__SNDC (command, n)

Здесь второй параметр n не является обязательным. Если он отсутствует или равен нулю, то command пересылается во все студенческие компьютеры и там выполняется. В противном случае пересылка организуется только в *P* с номером n.

ПРИМЕРЫ 16

```
A.   чистка экрана в комп'ютере 10
        _SNDC("CLS",10)
B.   удаление строки-подсказки  в
     комп'ютере 5
        _SNDC("KEY OFF",5)
```

Если в *P* из *T* передана некоторая команда, а там выполняется программа, то вычисления по ней прерываются и реализуется поступившая команда. Одновременно в __SNDC можно посылать несколько команд. Между ними в качестве разделителя требуется размещать символ ":" или код CHR$ (13).

ПРИМЕРЫ 17

```
A.   10 A$="FOR S=1 TO 99:PRINT S;:NEXT"
     20 C$="STOP"
     30 _SNDC(A$+CHR$(13)+C$,8)
B.   10 A$="FOR S=1 TO 99:PRINT S;:NEXT"
     20 C$="STOP"
     30 _SNDC(A$+":"+C$,8)
```

Общая «длина команд» в __SNDC не может превосходить 38 символов. Поэтому в некоторых случаях приходится использовать подряд несколько команд __SNDC. Однако это может привести к сбойным ситуациям. Пусть, например, из *T* требуется выполнить такие действия. Записать программу, находящуюся в ОЗУ компьютера 3, в виртуальную память (на диск mem) под именем "MOPE", а затем переслать в 3 программу с диска под именем "OKEAH". В примере 18 приведены два «решения» этой задачи. Первое из них несостоятельное. Здесь пересылка файла "OKEAH" начинается до окончания записи в mem файла "MOPE" (если последний не очень мал). Во втором случае этого не происходит. Там организован периодический опрос состояния в компьютере 3 ячейки &HF16E, «сигнализирующей» значениями 2 и 0 соответственно о продолжении или завершении операции SAVE "mem:MOPE".

ПРИМЕРЫ 18

```
A.   10 K$=CHR$(34)        'кавычки
     20 V$=CHR$(13)        'возврат каретки
     30 _SNDC(" SAVE"+K$+"mem:MOPE"+K$+V$,3)
     40 _SEND("OKEAH",3)
```

```
B.  10 K$=CHR$(34):V$=CHR$(13)
    20 _SNDC("SAVE"+K$+"mem:MOPE"+K$+V$,3)
    30 FOR S=1 TO 200:NEXT
    40 _PEEK(X,&HF16E,3,N)
    50 PRINT X;:IF X=2 THEN 30
    60 _SEND("OKEAH",3)
```

По __SNDC можно посылать из *T* в *P* и любые сетевые команды, кроме __ONLINE, доступные для *P* в данный момент времени.

Пример 19. Отключение из *T* студенческого компьютера номер 7 от локальной сети.

```
_SNDC("_OFFLINE",7)
```

Команды запуска программ и их останова можно передавать из *T* в *P* непосредственно, без __SNDC.

По команде __RUN (n, m) программа из ОЗУ студенческого компьютера с номером n запускается на счет со строки m. Если программа уже выполнялась, то она прерывается и заново запускается со строки m. Оба параметра в __RUN не обязательны. При отсутствии m программа запускается со строки с наименьшим номером. При отсутствии n или n=0 запускаются программы во всех *P*.

ПРИМЕРЫ 20

```
A.  _RUN(3,20)        B.  _RUN(0,100)
```

По команде __STOP(n) на компьютере с номером n производятся действия, аналогичные тем, которые вызываются нажатием на клавиатуре клавиш CTRL — STOP, то есть выполнение программы прерывается, вывод текста по LIST прекращается и т. п. Если параметр n не указан, то действие __STOP распространяется на все *P*.

ПРИМЕРЫ 21

```
A.  _STOP(5)    B.  _STOP
```

Почтовые ящики. В сетевом ОЗУ каждого компьютера имеется 2 специально выделенных участка памяти объемом по 256 байт, называемых соответственно передающим (SE) и принимающим (RE) почтовыми ящиками. Их начальные адреса можно получать так.

1. Для передающего ящика (SE):

PEEK (X, &H7C84):__PEEK (Y, &H7C85): A1 = X + 256 · Y

A1 = &H7900

2. Для принимающего ящика (RE):

__PEEK (X, &H7C86):__PEEK (Y, &H7C87): A2 = X + 256 · Y

A2 = &H7B00

С компьютера *T* обмен содержимым почтовых ящиков *T* и *P* осуществляется по командам __SNDMAIL и __RCVMAIL (*mail* — почта).

По команде __SNDMAIL (n) содержимое передающего ящика SE компьютера *T* пересылается в принимающий ящик RE студенческого компьютера с номером n. Если n не задан, то пересылка осуществляется во все *P*.

ПРИМЕРЫ 22

```
A.  _SNDM                 - ПЕРЕСЫЛКА SE ИЗ T
                            ВО ВСЕ P
B.  10 FOR K=4 TO 7       - ПЕРЕСЫЛКА SE ИЗ T
    20 _SNDM(K)             В КОМП'ЮТЕРЫ  4 - 7
    30 NEXT
```

Отметим, что специальных средств для формирования и чтения содержимого почтовых ящиков нет. Делать это можно, например, побайтно, с помощью команд __POKE и функции __PEEK (см. примеры 23 и 24).

Пример 23. Формирование содержимого передающего почтового ящика SE на *T* и его передача в компьютер 5.

```
10 A1$="Век живи  -  век учись! "
20 A2$="и ты наконец достигнешь "
30 A3$="того, что, подобно мудрецу, "
40 A4$="будешь иметь право сказать, "
50 A5$="что ничего не знаешь. "
60 A6$="КОЗЬМА ПРУТКОВ И"
70 A$=A1$+A2$+A3$+A4$+A5$+A6$
80 _PEEK(X,&H7C84):_PEEK(Y,&H7C85)
90 A1=X+256*Y                  ' A1=&H7900
100 FOR S=1 TO LEN(A$)
110 _POKE(ASC(MID$(A$,S,1)),A1+S-1)
120 NEXT
130 _SNDMAIL(5)
```

Пример 24. Распаковка содержимого принимающего почтового ящика (RE) на студенческом компьютере 5.

```
10 _PEEK(X,&H7C86):_PEEK(Y,&H7C87)
20 A2=X+256*Y                    '&h7B00
30 A$="":S=A2
40 _PEEK(R,S):R$=CHR$(R)
50 IF R$<>"" THEN A$=A$+R$:S=S+1:GOTO 40
60 PRINT A$
```

По команде __RCVMAIL(n) содержимое передающего ящика SE студенческого компьютера с номером n записывается в принимающий почтовый ящик RE компьютера *T* (см. примеры 25 и 26).

В примере 25 показано формирование содержимого передающего ящика SE в компьютере 6.

```
10 ' целые неотрицательные числа в ящик
20 DATA 7:                 'количество чисел
30 DATA   260, 1, 700, 3, 150, 684, 900
40 A1=&h7900:READ K:RESTORE 20
50 FOR S=0 TO K:READ R:A=A1+2*S
60 _POKE(RMOD256,A):_POKE(R\256,A+1)
70 NEXT
```

В примере 26 иллюстрируется прием на *T* в почтовый ящик RE содержимого передающего почтового ящика SE с компьютера 6 и распаковка полученных данных.

```
10 _RCVMAIL(6)
20 A2=&H7B00:_PEEK(U,A2):_PEEK(T,A2+1)
30 K=U+256*T
40 FOR S=1 TO K:A=A2+2*S
50 _PEEK(U,A):_PEEK(T,A+1)
60 R=U+256*T:PRINT R;
70 NEXT
```

Передача сообщений. Обмен короткими сообщениями между компьютерами можно проводить с помощью команд __MESSAGE и __TALK. Сообщения индицируются на последней строке экрана и не должны превышать 56 символов и установленной ширины текстового

экрана. Если принимающий компьютер находится в графическом режиме, то переданное ему сообщение будет выведено на экран лишь после перевода его в текстовый режим (SCREEN 0 или SCREEN 1).

По команде __MESSAGE (mes, n) значение строкового выражения mes в качестве сообщения пересылается в студенческий компьютер с номером n. Если n отсутствует, то сообщение передается всем подключенным к сети компьютерам *P*.

ПРИМЕРЫ 27

```
A.      _MESS("ПРЕКРАЩАЙТЕ РАБОТУ !")

B.      _MESS("РЕШАЙТЕ ЗАДАЧУ 3",11)
```

По команде __TALK (mes, v) (25)

значение строкового выражения mes передается в качестве сообщения из любого студенческого компьютера n. Если передача проходит успешно, то числовая переменная v принимает значение 0, если нет — 255. Рассмотрим два возможных случая.

1. n находится в состоянии __DISCOM.

Передача сообщения возможна только учительскому компьютеру *T*. Переменная v в (25) должна иметь значение 0.

ПРИМЕР 28

```
V=0:X$="МОЖНО ВЫЙТИ ?":_TALK(X$,V)
```

2. n находится в состоянии __ENACOM.

Передача сообщения возможна любому компьютеру, подключенному к сети. Переменная v должна иметь значение, равное номеру компьютера, которому сообщение передается.

ПРИМЕР 29

```
10 V=7:Y$="ПЕТЯ ! ПОМОГИ РЕШИТЬ ЗАДАЧУ"
20 _TALK(Y$,7)
```

Вот мы и ознакомились с довольно разнообразными и интересными средствами для работы с локальной вычислительной сетью в Бейсик-системе. Несмотря на простоту синтаксиса и прозрачность семантики рассмотренных команд и функций, их использование в реальных ситуациях в классе приводит к существенным потерям времени как учителя, так и студентов. Какой же отсюда следует вывод? Необходимо иметь удобные инструментальные средства для работы с локальной сетью.

ЧАСТЬ 2. АЛГОРИТМЫ

Применение математических методов не полезно, а вредно до тех пор, пока явление не освоено на доматематическом гуманитарном уровне.

Е.Вентцель

Во второй части пособия рассматриваются алгоритмы решения некоторых конкретных задач. В каждом случае дается полное описание алгоритма, а также или его обоснование, или обсуждение подходов к соответствующему обоснованию. Программы решения задач иллюстрируются многочисленными примерами. Выбор материала, на первый взгляд, может показаться весьма специфичным. Однако он продиктован отнюдь не случайными обстоятельствами, а отражает весьма широкий спектр возможных приложений приведенных алгоритмов и то незначительное внимание, которое отводится им в курсах численных методов по действующим программам естественнонаучных специальностей педагогических институтов.

Остановимся кратко на содержании данной части. Из методов внутренней сортировки подробно разобраны два наиболее интересных и эффективных по быстродействию и объему используемой дополнительной памяти алгоритма. Речь идет о так называемых пирамидальной и быстрой сортировках. Далее, на основе алгоритма генерирования перестановок решается задача коммивояжера при количестве пунктов $n \leqslant 10$. Симплекс-метод для общей задачи линейного программирования реализован двумя программами. При этом предварительного сведения задачи к канонической не требуется и ввод ограничений организуется весьма естественным способом. Из частных задач линейного программирования рассмотрены транспортная задача и задача о назначениях. Первая из них решается методом потенциалов, а для второй используется венгерский алгоритм. Наконец, обсуждаются три алгоритма приближенного вычисления корней многочленов с вещественными или комплексными коэффициентами. Стоит отметить, что последний из них — метод «обруча» — оказывается пригодным даже для нахождения корней большой кратности. Правда, в этом случае часто наблюдается потеря точности вычислений.

1. СОРТИРОВКА. ПОИСК. ПЕРЕБОР

1.1. Сортировка. Основные понятия

Пусть задан файл λ, состоящий из записей

$$\lambda(1), \lambda(2), \ldots, \lambda(n) \qquad (1)$$

Припишем каждой записи $\lambda(i)$ некоторый признак или, по-другому, ключ $K_{\lambda(i)}$ $(i=1, n)$. Обычно ключ — это какое-либо отдельное поле или комбинация полей записи. Будем считать, что на множестве ключей задано отношение линейного порядка или следования с обычными свойствами. Иными словами, элементы этого множества можно выстроить в порядке неубывания (невозрастания).

Задача сортировки файла ставится так. Найти такую перестановку записей (1):

$$\mu(1), \mu(2), \ldots, \mu(n) \qquad (2)$$

чтобы их ключи располагались в неубывающем порядке:

$$K_{\mu(1)} \leqslant K_{\mu(2)} \leqslant \ldots \leqslant K_{\mu(n)}.$$

При решении научных и технических задач довольно часто запись целиком составляет поле ключа.

Для получения из (1) файла (2), вообще говоря, требуется физическое перемещение записей в памяти ЭВМ. Во многих случаях нет надобности реально иметь перестановку (2). Достаточно описать ее тем или иным способом так, чтобы обеспечить непосредственный доступ к записям (1) в соответствии с порядком следования их ключей. Сделать это можно по-разному, например составлением списка адресов (индексов) элементов (2) в файле (1). Построение для (1) такого списка называют или адресным кодированием, или адресной сортировкой. Реализуется она просто, но требует дополнительной памяти. Часто адресное кодирование применяется в качестве первого этапа для обычной сортировки, то есть получения перестановки (2).

Пример 1. В таблице 1 приведен файл из 7 записей. Каждая из них представляет собой один элемент символьного массива и относится к конкретному студенту, имеющему задолженность по какому-либо предмету. Запись состоит из 5 отдельных полей: номер, фамилия и инициалы, курс, предмет, тип задолженности. Адресное кодирование данного файла в алфавитном порядке фамилий студентов осуществляется построением списка:

$$3, 5, 6, 7, 2, 4, 1 \qquad (3)$$

Таблица 1

Файл «Задолжники»

Ф. И. О.	Курс	Предмет	Тип задолженности
1 Щукина Э.	2Д	алгебра	зачет
2 Паниковский М. С.	1Б	физика	экзамен
3 Бендер О. И.	3А	физика	зачет
4 Синицкая З. В.	1Е	геометрия	экзамен
5 Воробьянинов И. М.	4В	алгебра	экзамен
6 Востриков Ф.	1В	физика	зачет
7 Изнуренков А. В.	5Г	алгебра	зачет

Здесь 3 указывает на первую запись (Бендер О. И.), ..., 1 — на последнюю запись (Щукина Э.).

При необходимости список (3) можно использовать для реального перемещения соответствующих записей в файле «Задолжники».

Иногда конкретный файл приходится сортировать одновременно по нескольким ключам.

Традиционно различают внутреннюю сортировку, обрабатывающую данные, хранимые в оперативной памяти, и внешнюю сортировку, оперирующую с данными, расположенными на дисках. Проблемы оптимизации существенно различаются в обоих случаях. При внутренней сортировке стараются уменьшить число сравнений ключей и перемещений записей файла. Во внешней сортировке решающим фактором эффективности соответствующего алгоритма становится количество обращений к дисковым устройствам.

В дальнейшем мы будем вести речь только о внутренней сортировке и рассматривать файлы, представляющие собой числовые или символьные одномерные массивы. Записями и ключами таких файлов будем считать значения соответствующих элементов массивов. Это не является каким-либо существенным ограничением. Дело в том, что обсуждаемые алгоритмы легко переносятся и на более общие случаи внутренних сортировок.

Пример 2. Отсортировать числовой массив:

$$7.2,\ 3,\ 8,\ 4,\ 8,\ 5.14,\ 9,\ 1. \qquad (4)$$

Обычная сортировка (4): 1, 3, 4, 5.14, 7.2, 8, 8, 9.

Адресная сортировка (4):

7	2,	3,	8,	4,	8,	5.14,	9,	1
5,	2,	2,	3,	7,		4,	8,	1

Имеется много разных алгоритмов сортировок. Самый простой и популярный из них — это «пузырьковая сортировка». Название ее происходит от образной интерпретации, при которой в процессе выполнения алгоритма более «легкие» элементы мало-помалу всплывают на «поверхность».

Пусть S — числовой массив

$$a(1),\ a(2),\ \ldots\ a(n)$$

Говорят, что элементы $a(i)$ и $a(j)$ из S образуют инверсию, если

$$i<j \text{ и } a(i)>a(j).$$

Алгоритм «пузырьковой сортировки» состоит в последовательных просмотрах снизу вверх (от конца к началу) массива S и обмене местами соседних элементов с инверсией. Более точно его можно понять из программы примера 3. Трудоемкость рассматриваемого алгоритма $O(n^2)$, то есть при любых S и n он решает задачу за $C \cdot n^2$ операций сравнений и обменов, где C — константа, от n не зависящая.

Пример 3. «Пузырьковая сортировка»


```
10 '          ПУЗЫРЬКОВАЯ СОРТИРОВКА
20 '=========================================
30 SCREEN 0:COLOR 15,4:KEY OFF
40 INPUT "количество элементов ";N: DIM A(N)
50 FOR I=1 TON:A(I)=INT(RND(-TIME)*100):NEXT
60 '-----------------------------------ЯДРО
70 FOR I=2 TO N:FOR J=N TO I STEP -1
80 IF A(J-1)>A(J) THEN SWAP A(J-1),A(J)
90 NEXT J,I
100 '---------------------------------------
110 FOR I=1 TO N:PRINT A(I);:NEXT I
120 END:'----трудоемкость алгоритма - O(n*n)
```

По данной программе по введенному числу N создается случайный массив из N целых чисел со значениями от 0 до 99, организуется его сортировка и последующая индикация. Ядром алгоритма являются строки 60—100.

«Пузырьковая сортировка» не требует для реализации дополнительной памяти. Однако из-за плохих временных характеристик она имеет лишь исторический интерес и вряд ли может быть рекомендована для практического использования. В следующих пунктах будут приведены два весьма эффективных алгоритма: «Пирамидальная сортировка» и «Быстрая сортировка». Первый из них имеет трудоемкость $O(n \cdot \ln n)$. Что касается второго алгоритма, то оценка трудоемкости у него не такая отличная, но большинство реальных массивов он сортирует быстрее первого.

1.2. Сортировка с помощью дерева

Все методы сортировок основаны на многократных просмотрах массивов S и выполнении определенных операций над их элементами. Как можно улучшить какой-либо из конкретных алгоритмов? По-видимому, путь один — за каждый просмотр всего S или его части получать как можно больше информации. Одна из возможностей на этом пути открывается при представлении сортируемого массива S в виде нелинейной структуры типа двоичного (бинарного) дерева D. На рисунке 30 показан такой граф D для S из n элементов, где $8 \leqslant n < 16$. В кружках размещены элементы массива:

9, 14, 8, 1, 5, 4, 9, 12, 3, 17, 1, 3

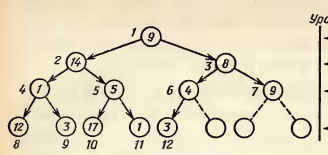


Рис. 30

Натуральными числами начиная от 1 и далее сверху вниз по уровням и слева направо на одном уровне пронумерованы все вершины D. Эти номера (адреса) проставлены около вершин. Именно такая адресация и будет использоваться в дальнейшем.

У бинарного дерева имеется, и притом только один, корневой узел, для которого нет предшественников (родителей). Адрес корня — 1. Любой другой узел имеет только одного предшественника и одного или двух преемников (потомков).

Иногда деревья изображают в виде двумерного массива с явным указанием адресов родителей и потомков для каждого узла $k\,(k=\overline{1,\ n})$. Однако в реальных ситуациях эти адреса проще не хранить, а вычислять по номеру k по таким формулам:

а) Для родителей $x(k)=k\backslash 2,\ k=1,\ 2,\ \ldots,\ n.$

б) Для потомка слева $y(k)=\begin{cases} 2*k, & k=1,\ 2,\ \ldots,\ n\backslash 2 \\ 0, & k>n\backslash 2 \end{cases}$.

в) Для потомка справа

$$z(k)=\begin{cases} 2*k+1, & k=1,\ 2,\ \ldots,\ (n-1)\backslash 2 \\ 0, & k>(n-1)\backslash 2 \end{cases}.$$

Заметим, что любой узел в дереве может быть корнем другого дерева — поддерева, именуемого номером узла, выбранного в качестве его корня.

Описанное представление массивов S в форме бинарных деревьев D будет использовано в следующем пункте. Сейчас же мы остановимся на одном родственном представлении S и некоторой дополнительной для S информации в форме дерева. Это позволит понять суть целой серии так называемых бинарных, или, по-другому, турнирных, сортировок. Пусть элементы S размещены в узлах нижнего уровня D (см. рис. 31). Тогда отсортировать S можно в два этапа, называемые соответственно «построением исходного дерева выбора» и «непосредственной сортировкой». Опишем их.

Этап 1. На первом шаге с помощью $n\backslash 2$ сравнений определяем наименьший из каждой пары элементов S с общим родителем. На втором шаге за не более чем $n\backslash 4+1$ сравнений находим наименьший элемент для пар уже отобранных ранее значений и т. д. В общем случае построение дерева выбора, корнем которого окажется наименьший в массиве ключ, потребует всего $n-1$ сравнений.

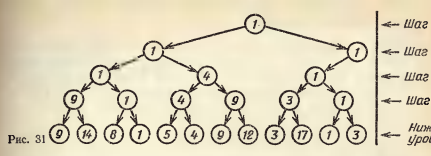


Рис. 31

Этап 2. Объявляем число α в корне дерева очередным элементом отсортированного массива и спускаемся в D по пути, им обозначенному. При наличии нескольких таких путей при движении сверху вниз предпочтение будем отдавать тем из них, которые включают потомков слева. Исключаем α из нижнего уровня D, заменяя его на $+\infty$, и проводим дополнительное сравнение элементов для пройденной ветви. При этом в корень D «всплывет» следующий наименьший элемент и т. д. Проход по ветви для отыскания очередного элемента отсортированного массива изображен на рисунке 32.

После n-кратного выполнения этапа 2 дерево D будет состоять из значений $+\infty$ и процесс сортировки закончится.

Для однократного выполнения второго этапа требуется выполнить $\log_2 n$ сравнений. Поэтому общая трудоемкость описываемых сортировок составляет $O(n\cdot\log_2 n)$ операций.

Отдельные алгоритмы турнирных сортировок строятся на основе изложенного метода и отличаются друг от друга способом представления исходной и сопутствующей информации, объемом дополнительно используемой памяти и т. п. При плохой организации вычислений трудоемкость оказывается хуже, чем $O(n\cdot\log_2 n)$.

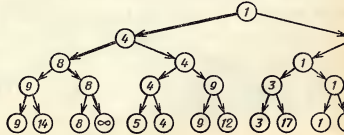


Рис. 32

1.3 Пирамидальная сортировка

Алгоритм сортировки массивов S, предложенный Дж. Уильямсом и развитый Р. Флойдом, носит название «пирамидального». Он базируется на представлении S в виде бинарного дерева D, не требует дополнительной памяти и имеет трудоемкость $O(n \cdot \log_2 n)$. Об этом алгоритме и пойдет речь дальше. Начнем с определения.

Пусть задан массив

$$S(1),\ S(2),\ \ldots,\ S(n) \qquad (5)$$

Пирамидой называется непустая последовательность элементов (5) вида

$$S(p),\ S(p+1),\ \ldots,\ S(q)\ (1 \leqslant p \leqslant q \leqslant n) \qquad (6)$$

для которой выполнено одно из следующих условий.

1) $2p > q$
2) $2p = q$ и $S(p) \geqslant S(q)$
3) $2p < q$ и

$$S(j) \geqslant S(2j)\ (p \leqslant j \leqslant q/2)$$
$$S(j) \geqslant S(2j+1)\ (p \leqslant j \leqslant (q-1)/2)$$

Из определения немедленно вытекают такие утверждения:

1. Для любой последовательности (5) подпоследовательность

$$S(n\backslash 2+1),\ S(n\backslash 2+2),\ \ldots,\ S(n)$$

является пирамидой.

2. Если последовательность (5) — пирамида, то

$$S(1) \geqslant \max_{1 \leqslant j \leqslant n} S(j) \qquad (7)$$

3. Если последовательность (5) — пирамида и представлена в виде бинарного дерева D, то значение любого узла в D будет не меньше значений его левого и правого потомков.

Пример 4. Последовательность

$$90,\ 70,\ 11,\ 8,\ 3,\ 9,\ 7,\ 5,\ 6,\ 1,\ 2$$

является пирамидой. Наглядно это видно из рисунка 33.

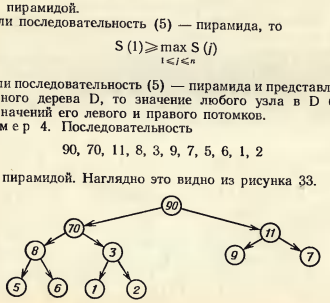


Рис 33

Теперь можно приступить к описанию алгоритма пирамидальной сортировки. Состоит она в реализации двух этапов. Остановимся на них подробнее.

Этап 1. **Построение пирамиды.** В (5) «хвост», т. е. подпоследовательность

$$S(n\backslash 2+1),\ S(n\backslash 2+2),\ \ldots,\ S(n) \qquad (8)$$

как мы уже отмечали, является пирамидой. Будем наращивать (8), добавляя к ней оставшиеся элементы из (5). Пусть $S(j+1)$, $S(j+2)$, ..., $S(n)$ — пирамида. Припишем к ней слева элемент $S(j)$:

$$S(j),\ S(j+1),\ \ldots,\ S(n) \qquad (9)$$

и преобразуем (9) снова в пирамиду. Для этого «просеем» $S(j)$ по соответствующей веточке двоичного представления (5). Делается это так. Рассмотрим $S(j)$ и два его потомка $S(2j)$ и $S(2j+1)$. Если $S(j)$ не меньше потомков, то вычисления прекращаем, ибо (9) — уже пирамида. В противном случае обмениваем значения

$$S(j) \text{ и } \max(S(2j),\ S(2j+1))$$

в соответствующих позициях D и «опустившийся» элемент продолжаем просеивать тем же самым способом. В конце концов преобразованная последовательность (9) станет пирамидой.

Продолжая наращивание (9), мы превратим весь массив (5) в пирамиду. При этом окажется выполненным неравенство (7).

Объявляем полученную пирамиду S текущей и переходим к следующему этапу.

Этап 2. **Сортировка пирамиды.** В текущей пирамиде S первый элемент не меньше остальных. Поступим с ним так. Обменяем значениями концевые элементы S и укоротим S справа на 1. Полученная последовательность уже может и не быть пирамидой Применим к ней процесс «просеивания» для элемента $S(1)$, описанный на этапе 1. Преобразованная последовательность S становится пирамидой.

Повторяя этап 2 ($n-1$) раз, отсортируем S по невозрастанию

Отдельные этапы проведения «пирамидальной сортировки» наглядно можно проследить на конкретном примере, прогоняя его или на бинарном дереве D, или непосредственно выписывая текущи последовательности S.

Пример 5. Для последовательности 23, 77, 12, 7, 44, 82, 16, 53 произвести «пирамидальную сортировку», выписывая текущие значения S в процессе выполнения алгоритма.

В строках таблицы 2 показаны требуемые значения S после каждой реализации этапов 1 и 2. При этом в первом случае ступенчатая линия отделяет остаток последовательности от наращиваемой слева пирамиды, а во втором — сокращаемую справа последовательность перед просеиванием ее первого элемента от формируемого с конца отсортированного массива.

Таблица 2. Построение пирамиды и ее сортировка

построение пирамиды							
23	77	12	7	44	82	16	53
23	77	12	53	44	82	16	7
23	77	82	53	44	12	16	7
23	77	82	53	44	12	16	7
82	77	23	53	44	12	16	7
сортировка пирамиды							
7	77	23	53	44	12	16	82
16	53	23	7	44	12	77	82
12	44	23	7	16	53	77	82
12	16	23	7	44	53	77	82
7	16	12	23	44	53	77	82
12	7	16	23	44	53	77	82
7	12	16	23	44	53	77	82

Анализ «пирамидальной сортировки» показывает, что для ее выполнения требуется не более $3n \log_2 n$ элементарных операций типа сравнений и обменов. Подобная эффективность и гарантированная надежность для худшего случая часто оказываются решающими факторами, заставляющими в конкретных ситуациях отдавать предпочтение данному способу сортировки.

В примере 6 приводится программа для «пирамидальной сортировки» одномерных массивов по неубыванию. Для сортировок по невозрастанию в строках 320 и 330 необходимо знаки отношений "<" и ">=" заменить соответственно на ">" и на "<=".

Ядром программы являются строки 250—350. Строка 260 организует построение пирамиды, а строки 270—290 — ее сортировку. На обоих этапах вычислений используется подпрограмма «просеивания», расположенная в строках 310—350. Некоторые временные характеристики программы при прогонке примеров на ЭВМ «Ямаха» указаны в строках 120—160. Например, массив из 2000 элементов сортируется менее чем за 11 минут. Учитывая весьма медленное быстродействие используемого компьютера, следует признать это время вполне приемлемым.

Пример 6. Программа «пирамидальной сортировки»

```
10 '          ПИРАМИДАЛЬНАЯ СОРТИРОВКА
20 '-----------------------------------------------
30 '          ХАРАКТЕРИСТИКИ ПРОГРАММЫ
40 '-----------------------------------------------
50 '   По программе "ПИРАМИДА" проводятся
60 'сортировки  одномерных массивов по неу-
70 'быванию. Ключами являются элементы мас-
80 'сива. Для  сортировок по невозрастанию
90 'необходимо в строках 320 и 330 знаки от-
100 'ношений заменить соответств. на > и <=.
110 ' Общее количество операций- O(N*LN(N)).
120 ' Время сортировки T:
130 '  N=100 , T=          19.0 сек
140 '  N=500 , T=   2 мин  8.0 сек
150 '  N=1000, T=   4 мин 47.7 сек
160 '  N=2000, T=  10 мин 37.1 сек
```

```
170 '-----------------------------------ВВОД
180 SCREEN 0:COLOR 15,4:KEY OFF
185 DEFINT N,K,X,Y,J
190 PRINT "ПИРАМИДАЛЬНАЯ  СОРТИРОВКА"
200 PRINT "--------------------------"
210 INPUT "количество элементов ";N:DIM S(N)
220 CLS
230 LOCATE 8,9:PRINT "проводятся вычисления"
240 GOSUB 430
250 TIME=0:K=N:'== ====================ЯДРО
260 FOR J=N\2 TO 1 STEP-1:X=J:GOSUB 310:NEXT
270 FOR K=N-1 TO 1 STEP -1
280 SWAP S(1),S(K+1):X=1:GOSUB 310
290 NEXT:GOTO 360
300 '---------------подпрограмма просеивания
310 Y=X+X:ON SGN(Y-K)+2 GOTO 320,330,350
320 IF S(Y)<S(Y+1) THEN Y=Y+1
330 IF S(X)>=S(Y) THEN 350
340 SWAP S(X),S(Y):X=Y:GOTO 310
350 RETURN:'=========================ВЫВОД
360 CLS
370 PRINT "время сортировки -";TIME/50;"сек"
380 PRINT "количество элементов -";N:PRINT
390 PRINT "массив:";
400 FOR J=1 TO N:PRINT S(J);:NEXT J
410 END:'-----------имя программы "ПИРАМИДА"
420 '          ввод или генерирование массива
430 FOR J=1 TO N:S(J)=INT(RND(1)*100):NEXT J
440 RETURN
```

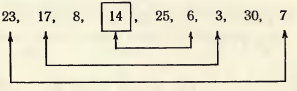

1.4. Быстрая сортировка

При «пузырьковой сортировке» производились обмены ключей в соседних позициях. При «пирамидальной сортировке» такой обмен совершался между ключами в позициях, жестко связанных друг с другом соответствующим бинарным деревом. Здесь будет рассмотрен алгоритм сортировки К. Хоора, использующий несколько иной механизм выбора значений для обменов. Этот алгоритм называется или сортировкой с разделением, или «быстрой сортировкой». В первом случае подчеркивается положенный в его основу метод последовательного дробления массива на части, а во втором — неожиданная эффективность. В подавляющем большинстве реальных случаев использование «быстрой сортировки» дает лучшие временные результаты по сравнению с другими методами. Однако следует иметь в виду, что для нее нет гарантированной трудоемкости типа $O(n \cdot \log_2 n)$, где n — количество элементов массива. Более того, в иных ситуациях трудоемкость достигает $O(n^2)$ и не может быть снижена. Иллюстрацией к этому могут служить упорядоченные массивы. Исходя из сказанного, в особо критических ситуациях, где результат должен выдаваться в жестко очерченном интервале времени, применять «быструю сортировку» следует весьма осмотрительно. По образному выражению Н. Вирта [18], «...быстрая сортировка напоминает азартную игру, где следует заранее рассчитать, сколько можно позволить себе проиграть в случае невезения».

Перейдем к сути метода. Пусть $S(k)\,(k=\overline{1,\ n})$ — одномерный массив и x — какой-либо элемент из $S(k)$. Сначала обсудим, как разбить S на две непустые непересекающиеся части S_1 и $S_2\,(S_1 \cup S_2 = S)$ так, чтобы в S_1 оказались элементы, не превосходящие x, а в S_2 — не меньшие x:

$$a \leqslant x,\ a \in S_1 \qquad b \geqslant x,\ b \in S_2.$$

Начнем с примера. Пусть в массиве S:



зафиксирован элемент $x=14$. Просматриваем S слева направо, пока не найдем $a>x$. Получим $a=23$. Далее, просматриваем S справа налево, пока не найдем $b<x$. Получаем $b=7$. Меняем местами a и b. Продолжая этот процесс, придем к последовательности

6, 7, 3, 8, 6 | 25, 14, 17, 30, 23

разделенной на две требуемые части S_1 и S_2.

В примере 7 показан алгоритм разделения для общего случая.

Пример 7. Алгоритм разделения

```
10 '  ввод  N ; S(K), K=1,2,...N ; X
20 '-----------------------------------------
30 I=1:J=N
40 IF S(I)<X THEN I=I+1:GOTO 40
50 IF X<S(J) THEN J=J-1:GOTO 50
60 IF I<=J THEN SWAP S(I),S(J):I=I+1:J=J-1
70 IF I<=J THEN 40
80 IF I<=N THEN P=I-1 ELSE P=N-1
90 '-----------------------------------------
100 ' вывод S(K), K=1,2,...N ; P
110 ' P - позиция разделения
```

Через P обозначена позиция разделения S_1 и S_2. Элементы от 1 до P относятся к S_1, а остальные — к S_2.

В дальнейшем значение x для разделения S мы будем выбирать из «середины» массива. Если количество элементов в S четно, то есть имеются два кандидата на x, то условимся использовать из них тот, который имеет меньший индекс.

Теперь до алгоритма «быстрой сортировки» остается сделать один незначительный шаг. Разобьем S на две части S_1 и S_2 так, как это описано ранее. Аналогично поступим с S_1 и S_2 и так далее, пока каждая из частей не будет содержать ровно 1 элемент. Легко понять, что в результате получится отсортированный массив. При реализации этой процедуры на ЭВМ остается позаботиться лишь о хранении адресов раздваиваемых подмассивов. Если каждый раз продолжать работу с меньшей из полученных частей, отправляя адреса начала и конца второй части на хранение, то потребуется совсем немного дополнительной памяти. Одновременно придется помнить не более $2\log_2 n$ чисел (адресов). Скажем, при $n \leqslant 4000$, $\log_2 n \leqslant 12$.

В примере 8 приведена программа «Быстрая», реализующая описанный алгоритм Хоора. В ней адреса подмассивов хранятся в виде наращиваемого стека, выборка из которого производится в порядке, обратном занесению. Иными словами, последний подмассив, обозначенный в стеке, обрабатывается первым. В строках 140—180 программы указано время сортировки по ней случайных контрольных массивов при разных n. Интересно провести сопоставление этих цифр с соответствующими данными для «пирамидальной сортировки».

Пример 8. Программа быстрой сортировки Хоора


```
10 '        Б Ы С Т Р А Я   СОРТИРОВКА
20 '-----------------------------------------
30 '          ХАРАКТЕРИСТИКИ ПРОГРАММЫ
40 '-----------------------------------------
50 '  По программе "БЫСТРАЯ" проводятся сор-
60 ' тировки одномерных массивов по неубыва-
70 ' нию. Ключами являются элементы массива.
80 ' Для сортировок по невозрастанию необхо-
90 ' димо в строках 310 и 320 знак отношения
100 ' "<" заменить знаком ">".
110 ' Дополнительная память :
120 '   СТЕК [T1(13),T2(13)]
130 ' Время сортировки случайного массива:
140 '   N=100   ,  T=          16.0 сек
150 '   N=500   ,  T=   1 мин 31.2 сек
160 '   N=1000  ,  T=   3 мин 14.2 сек
170 '   N=2000  ,  T=   6 мин 34.9 сек
180 '   N=3000  ,  T=  10 мин 18.7 сек
190 '-----------------------------------------
200 SCREEN 0:COLOR 15,4:KEY OFF
210 DEFINT I,J,A,B,K,T,N
220 PRINT "Б Ы С Т Р А Я   СОРТИРОВКА ХООРА"
230 PRINT "----------------------------------"
240 INPUT "количество элементов ";N:CLS
250 LOCATE 8,9:PRINT "проводятся вычисления"
260 DIM S(N),T1(13),T2(13):GOSUB 480
270 TIME=0:'==========================ЯДРО
280 K=1:T1(1)=1:T2(1)=N  :'-------иниц. стека
290 A=T1(K):B=T2(K):K=K-1:'--чтение из стека
300 I=A:J=B:X=S((A+B)\2) :'--------разделение
```

```
310 IF S(I)<X THEN I=I+1:GOTO 310
320 IF X<S(J) THEN J=J-1:GOTO 320
330 IF I<=J THEN SWAP S(I),S(J):I=I+1:J=J-1
340 IF I<=J THEN 310
350 IF J-A>=B-I THEN 380:'-----запись в стек
360 IF I<B THEN K=K+1:T1(K)=I:T2(K)=B
370 B=J:IF A<B THEN 300 ELSE 400
380 IF A<J THEN K=K+1:T1(K)=A:T2(K)=J
390 A=I:IF A<B THEN 300
400 IF K>0 THEN 290:'==================ВЫВОД
410 CLS:PRINT "время сортировки  - ";TIME/50
420 PRINT "количество элементов - ";N
430 PRINT "--------------------------------"
440 PRINT "массив:";
450 FOR U=1 TO N:PRINT S(U);:NEXT U
460 END:'-------------имя программы "БЫСТРАЯ"
470 '--------- ввод или генерирование массива
480 FOR J=1 TO N:S(J)=INT(RND(1)*1000): NEXT
490 RETURN:'-------------------------------
```

1.5. Поиск. Осиовные понятия

С задачами поиска требуемой информации человеку приходится сталкиваться постоянно. Типичными примерами могут служить работа с тем или иным справочником, телефонной книгой, картотекой в библиотеке или распространенная до недавнего времени практика вычислений с помощью специальных таблиц. В последнем случае поиск конкретного значения функции заменял собой весьма трудоемкий процесс его непосредственного вычисления.

Перейдем к более формальному разговору о задачах поиска. Будем предполагать, что файл λ состоит из записей

$$\lambda(1), \lambda(2), \ldots, \lambda(n)$$

с ключами $K_{\lambda(i)}$ $(i=\overline{1, n})$ и K — некоторое значение. Встречающиеся на практике разновидности простейших задач поиска могут быть сформулированы в одном из следующих четырех видов.

A. Требуется определить по крайней мере одну запись в λ, имеющую K своим ключом.

B. Все записи в λ, имеющие K своим ключом.

C. Если в λ нет записей с ключом K, то добавить в λ новую запись.

D. Включить в λ новую запись.

K называют аргументом поиска или запросом, задачи A и B — простым поиском, а задачи C и D — поиском со вставкой. Поиск по запросу K в задачах A и B может быть безрезультатным.

Иногда в задачах A — C поиск организуется не по совпадению с запросом K, а по выполнению некоторых условий для определенной функции от ключей. Примером может служить поиск по интервалу значений ключа, когда отыскиваются записи $\lambda(i)$ с условиями: $a \leqslant K_{\lambda(i)} \leqslant b$, где a и b — константы.

Методы поиска можно классифицировать по разным признакам: местонахождению λ, изменению λ, схеме доступа к отдельным записям $\lambda(i)$ $(i=\overline{1, n})$, способу сравнения ключей или функций от них и т. п. Например, поиск считается внутренним, если λ целиком размещается в оперативной памяти, и внешним — в противном случае. Далее, для задач A и B поиск называется статическим, для задач C и D — динамическим.

Мы будем заниматься только внутренним поиском. При этом ограничимся рассмотрением сформулированных задач, совершенно не затрагивая таких интересных обобщений, как поиск по запросам с искажениями. В дальнейшем под файлом будем понимать одномерный или двумерный числовой или символьный массив и, следовательно, вести поиск в одномерной или двумерной таблице.

Отметим, что различные варианты задач поиска встречаются при решении многих других проблем. Достаточно вспомнить ранее рассмотренные методы сортировок, где постоянно приходилось находить пары значений для текущих обменов. Впрочем, от правильной организации исходной информации во многом зависит эффективность алгоритмов поиска. И здесь уже сама сортировка может оказаться полезным предварительным преобразованием λ.

В различных АСУ поиск требует наибольших временных затрат. Поэтому качество соответствующего программного обеспечения находится в прямой зависимости от типа используемого в нем алгоритма поиска.

К просто реализуемым методам поиска относится так называемый последовательный поиск, при котором осуществляется просмотр подряд всех записей λ до получения решения задач A — C или их обобщений. Предоставим читателю самостоятельно написать соответствующие программы при последовательном поиске в упорядоченной или неупорядоченной одномерной таблице (линейном списке) и проанализировать их.

В примере 9 приведена программа одного из наиболее употребительных и эффективных методов поиска в упорядоченных массивах. Он имеет несколько названий: бинарный поиск, логариф-

мический поиск или метод деления пополам. Основная идея бинарного поиска довольно проста. Сначала K сравнивается со средним ключом в таблице. Результат сравнения или приводит к решению задачи, или позволяет определить, в какой части массива — верхней или нижней — продолжать поиск. Применяя описанный процесс к полученному «укороченному» наполовину массиву через не более чем $\log_2 n$ сравнений, получим позитивное или негативное решение.

Пример 9. Бинарный поиск в упорядоченном одномерном массиве

```
10 ' БИНАРНЫЙ ПОИСК ПО РАВЕНСТВУ (задача А)
20 ' МАССИВ S() ОТСОРТИРОВАН ПО НЕУБЫВАНИЮ
30 '-------------------------------------ВВОД
40 COLOR 15,1:SCREEN 0:KEY OFF
50 INPUT "количество элементов ";N:DIM S(N)
60 FOR I=1 TO N:READ S(I):NEXT I
70 DATA 2,7,8,9,13,15,16,17,18,23,24,89
80 INPUT "ключ                 ";K
90 A=1:B=N:'=============================ЯДРО
100 IF B>=A THEN I=(A+B)\2 ELSE 150
110 IF K<S(I) THEN B=I-1:GOTO 100
120 IF K>S(I) THEN A=I+1:GOTO 100
130 '=============================ВЫВОД
140 PRINT "I=";I,"S(I)=";K:GOTO 160
150 PRINT "неудача"
160 END:'-------------------------------------
```

1.6. Поиск в сети

Пусть задана ориентированная сеть $S=(P, R, d)$ (см. рис. 34), где P — множество вершин; R — множество дуг; d — функция, определенная на R со значениями в множестве действительных чисел.

Иногда $d(x; y)$ интерпретируют как длину дуги $(x, y)\in R$.

В различных алгоритмах оптимизации на сетях во внутренних циклах многократно приходится решать следующие две задачи.

А. Для фиксированной вершины $x\in P$ определить все дуги сети вида (x, y) (выходы из x).

В. Для фиксированной вершины $x\in P$ определить все дуги сети вида (y, x) (входы в x).

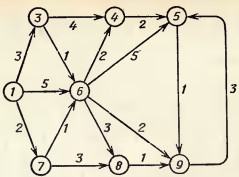


Рис. 34

Если находить дуги, выходящие из x или входящие в x, примитивной процедурой поиска с перебором всех элементов из R, то трудно ожидать получения качественных алгоритмов.

Здесь мы обсудим один из способов ускорения поиска в сети. Будем считать, что S представлена в оперативной памяти в виде массива дуг λ так, как это указано в таблице 3, и что записями λ являются его отдельные строки.

Пусть вначале решается задача А. Отсортируем файл λ по ключу x. Тогда ясно, что среднее время нахождения всех дуг (x, y) даже при последовательном поиске сократится примерно наполовину.

Таблица 3

	x	Дуги y	d		x	Дуги y	d
1	7	8	3	9	1	3	3
2	3	4	4	10	1	7	2
3	5	9	1	11	6	4	2
4	9	5	3	12	1	6	5
5	4	5	2	13	6	8	3
6	8	9	1	14	6	5	5
7	3	6	1	15	6	9	2
8	7	6	1				

Кроме того, сразу становится возможным организовать и более эффективный бинарный поиск. Но мы пойдем не по этому пути. Пусть имеется некоторый вспомогательный массив A, в позициях x которого записан адрес $A(x)$ расположения в λ начальной дуги вида (x, y). В этом случае проблема поиска всех дуг (x, y) сводится просто к выборке адреса $A(x)$. Если, например, требуется найти в λ все дуги вида $(6, y)$, то, получив $A(6)=8$, сразу же попадаем на требуемый подмассив выходов из 6 (см. рис. 35).

Описанный способ решения задачи A предполагает, что $x\in P$ — это натуральные числа. При этом для минимизации объема дополнительной памяти, т. е. величины $l=\max\limits_{x\in P} x$, желательно иметь последовательную нумерацию вершин начиная с 1.

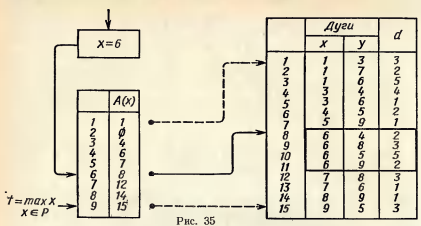


Рис. 35

Перейдем к задаче В. Если отсортировать λ по y, то удастся построить аналогичный простой механизм поиска входов в конкретную вершину. Однако при этом будет утеряна возможность эффективно решать задачу А. Поэтому поступим несколько иначе. Введем в рассмотрение два вспомогательных одномерных массива $B(x)$ и $C(x)$. Пусть $B(x)$ устроен так же, как и $A(x)$. В позиции x у $B(x)$ указывается адрес α первой от начала строки в λ с дугой (y, x). Далее, если по некоторому адресу α в λ записана дуга (y, x), то в $C(\alpha)$ указывается адрес в λ следующей такой дуги. Символ $\varnothing$ в $B(x)$ используется для указания отсутствия входов в вершину x, а в $C(x)$ — как метка завершения соответствующего подмассива входов. При такой организации $B(x)$ и $C(x)$ проблема поиска всех входов в x сводится к выборкам сначала адреса $B(x)$, а затем последовательных адресов из C. Наглядно описанный механизм поиска входов в конкретную вершину сети показан на рисунке 36.

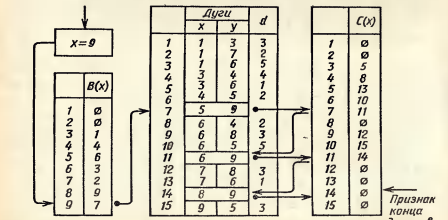


Рис. 36

Заметим, что формирование массивов B и C фактически приводит к адресному кодированию λ. Иными словами, осуществляется адресная сортировка λ по y. Более того, для рассмотренной ранее организации поиска выходов из конкретных вершин можно было бы отказаться от начальной сортировки λ по x, заменив ее подобным адресным кодированием. Правда, при этом пришлось бы к массиву A добавить еще один вспомогательный массив D, устроенный аналогично C и используемый для хранения «последующих» адресов.

Массивы A, B и C будем называть справочниками, а их совокупность — справочной для λ. Таким образом, по массиву λ нам необходимо научиться строить справочники: а) A б) B и C в) A; B и C.

Сделать это несложно. При этом в самом общем случае после сортировки все организуется за один просмотр λ (см. примеры 10—12).

Пример 10. Адреса выходов из x. Справочник А

```
10 SCREEN 0:COLOR 15,4:KEY OFF
20 INPUT "количество дуг сети ";N:DIM S(N,2)
30 '-------------------------------------------
40 '          ввод массива S()
50 '-------------------------------------------
60 INPUT "максимальный номер пункта ";M
70 DIM B(M),C(N):'====================ЯДРО
80 FOR I=N TO 1 STEP -1
90 C(I)=B(S(I,1)):B(S(I,1))=I
100 NEXT I:'=============================
```

Пример 11. Адреса входов в x. Справочники В и С

```
10 SCREEN 0:COLOR 15,4:KEY OFF
20 INPUT "количество дуг сети ";N:DIM S(N,2)
30 '-------------------------------------------
40 '      ввод и сортировка массива S()
50 '-------------------------------------------
60 INPUT "максимальный номер пункта ";M
70 DIM A(M):'=========================ЯДРО
80 FOR I=N TO 1 STEP -1:A(S(I,0))=I:NEXT I
90 '=========================================
```

Пример 12. Входы и выходы. Справочники A, B и C

```
10 SCREEN 0:COLOR 15,4:KEY OFF
20 INPUT "количество дуг сети ";N:DIM S(N,2)
30 '-------------------------------------
40 '    ввод и сортировка массива S()
50 '-------------------------------------
60 INPUT "максимальный номер пункта ";M
70 DIM A(M),B(M),C(N):'==============>ЯДРО
80 FOR I=N TO 1 STEP -1
90 C(I)=B(S(I,1)):A(S(I,0))=I:B(S(I,1))=I
100 NEXT I:'=================================
```

Указанная организация справочной эффективна при реализации потоковых алгоритмов. Мы на этом останавливаться не будем.

1.7. Генератор перестановок

Иногда задачи решают полным перебором возможных вариантов. И хотя этот прием, как правило, является весьма трудоемким по времени, идти на него приходится, если ничего лучшего придумать не удается. В подобных «переборных» задачах полезным бывает алгоритм получения всех $n!$ перестановок из элементов множества

$$S=\{a_1,\ a_2,\ ...,\ a_n\}, \qquad (10)$$

где $a_k\ (k=\overline{1,n})$ — попарно различные действительные числа.

Ниже приведен вариант алгоритма генерирования перестановок.

Шаг 1. Отсортируем (10) в порядке убывания. Полученную перестановку объявляем начальной.

Шаг 2. Находим наименьший индекс $i\in\{2,\ 3,\ ...,\ n\}$, при котором $a_{i-1}>a_i$.

Шаг 3. Находим наименьший индекс $j\in\{1,\ 2,\ ...,\ i-1\}$, при котором $a_j>a_i$.

Шаг 4. Обменниваем значениями a_i и a_j.

Шаг 5. Обменниваем значениями компоненты в парах:

$$(a_1,\ a_{i-1}),\ (a_2,\ a_{i-2}),\ ...,\ (a_k,\ a_{i-k})\ \left(k=\left[\frac{i-1}{2}\right]\right).$$

Шаг 6. Выводим очередную сформированную перестановку и если их общее количество меньше $n!$, то переходим к шагу 2. В противном случае вычисления прекращаем.

Покажем, что, используя приведенный алгоритм, мы получим все $n!$ различных перестановок. Не ограничивая общность, можно считать, что S состоит из последовательных натуральных чисел: 1, 2, ..., n. Зафиксируем $t>n$ и рассмотрим t-ричную систему счисления с цифрами 0, 1, 2, ..., n, ..., $t-1$.

В множестве B n-значных t-ричных чисел выделим подмножество B_0 чисел, у которых цифры попарно различны и являются элементами S. Перестановки из S и элементы B_0 можно поставить во взаимно однозначное соответствие по правилу:

$$(a_1,\ a_2,\ ...,\ a_n) \leftrightarrow a_n t^{n-1}+a_{n-1}t^{n-2}+...+a_2 t+a_1.$$

Следовательно, на проблему получения всех перестановок из S можно смотреть как на задачу выделения в B подмножества B_0. Пусть элементы B_0 выписаны в порядке возрастания: $y_1<y_2<...<y_{n!}$. Тогда шаг 1 алгоритма определяет первоначальную перестановку $(n,\ n-1,\ ...,\ 1)$, соответствующую числу

$$y_1=1\ 2\ 3\ ...\ n.$$

Если уже сгенерирована перестановка для $y_k\in B_0\ (k<n!)$, то совокупность шагов 3—6 как раз и формирует перестановку, соответствующую числу $y_{k+1}\in B_0$. Этим и доказывается, что будут получены все $n!$ перестановок из S.

Нелишне помнить, что функция $n!$ растет очень быстро. Например, вопрос о том, сколько списков можно составить из десяти различных фамилий, приводит уже к числу $10!=3628800$. Поэтому практическое использование программы «Генератор перестановок» (см. пример 13) ограничено значениями $n<10$.

Пример 13. Генератор перестановок

```
10 '              "ГЕНЕРАТОР ПЕРЕСТАНОВОК"
20 'ПОЛУЧЕНИЕ S! ПЕРЕСТАНОВОК ИЗ S ЭЛЕМЕНТОВ
30 '-------------------------------------------
40 ' характеристики быстродействия
50 '   S=5,  время                  6.1 сек
60 '   S=6,  время                 36.9 сек
70 '   S=7,  время  4 мин 19.2 сек
80 '-------------------------------------------
90 COLOR 15,4:SCREEN 0:KEY OFF:DEFINT A-S
100 INPUT "количество элементов ";S:DIM P(S)
110 FOR I=1 TO S:P(I)=S+1-I:NEXT
120 Z=1:FOR I=1 TO S:Z=Z*I: NEXT:'-------- S!
130 TIME=0:Y=1:GOSUB 250:'==============>ЯДРО
140 FOR Y=2 TO Z:I=2:J=1
150 IF P(I-1)<=P(I) THEN I=I+1:GOTO 150
160 IF P(J)<=P(I) THEN J=J+1:GOTO 160
170 SWAP P(I),P(J):IF I=2 THEN 190
```

```
180 FOR L=1 TO (I-1)/2:SWAP P(L),P(I-L):NEXT
190 GOSUB 250
200 NEXT Y
210 PRINT "время =";TIME/50:END
220 '=_================================
230 '          распечатка перестановки
240 '        или иное ее использование
250 PRINT Y;")   ";
260 FOR E=1 TO S:PRINT P(E);:NEXT:PRINT
270 RETURN:'-------------------------------
```

В качестве примера применения генератора перестановок можно было бы рассмотреть такую задачу из механики.

Заданные n масс m_k ($k=\overline{1, n}$) требуется распределить в пространстве, например в капсуле космического корабля, в точках M_1, M_2, ..., M_n так, чтобы их центр тяжести был максимально близок к фиксированной точке M_0.

Оставляя детальный разбор этой типично «переборной» задачи читателю, мы проиллюстрируем метод полного перебора на несколько иной проблеме. Речь пойдет о широко известной «задаче коммивояжера». Формулируется она так.

Коммивояжер (агент по сбыту), отправляясь из своего города, должен *кратчайшим маршрутом* посетить ровно по одному разу $n-1$ заданных городов и вернуться назад.

Эта оптимизационная задача и различные ее модификации в действительности возникают не только при доставке товаров на дом, но и в ситуациях совершенно иного характера [22]. Математические модели «задачи коммивояжера» содержат большое количество переменных и ограничений. Поэтому для их решения общие методы линейного программирования обычно не используются. Предпочтение отдается вычислительной схеме, известной под названием метода ветвей и границ (ветвления и ограничения) [22].

При небольших n мы попробуем справиться с задачей коммивояжера перебором всех замкнутых маршрутов, проходящих по одному разу через каждый из городов. Будем называть такие маршруты допустимыми. Пусть города пронумерованы числами: 1, 2, ..., n, расстояния между ними равны $m(i, j)(i, j=\overline{1, n})$, причем значения $m(i, j)$ и $m(j, i)$ не обязательно совпадают. При отсутствии дороги между пунктами i и j можно считать, что она есть, но $m(i, j)=\infty$.

Всякая перестановка $(a_1, a_2, ..., a_n)$ из элементов: 1, 2, ..., n определяет допустимый маршрут:

$$a_1 \to a_2 \to \ldots \to a_n \qquad (11)$$

Тогда для решения «задачи коммивояжера», перебирая все возможные перестановки (a_1, a_2, ..., a_n), достаточно вычислять длины соответствующих маршрутов и запоминать лучшие из них. Именно эта идея и реализована в программе КОММИ примера 14. Причем учитываются только перестановки (11), где $a_n=1$. Ввиду замкнутости (11) это делать можно. Таким образом, для n городов просматривается и анализируется $(n-1)!$ маршрутов.

Пример 14. Задача коммивояжера

```
10 '  ЗАДАЧА КОММИВОЯЖЕРА (полный перебор)
20 '--+-------------------------------ВВОД
30 SCREEN 0:COLOR 15,4:KEY OFF:DEFINT A-S
33 P$="оптимальные маршруты коммивояжера:"
40 INPUT "количество пунктов ";S
44 IF S<2 THEN 40
50 DIM M(S,S),Q(30,S),P(S):B=2^15-1
60 ' M(,)- матрица расстояний
70 ' Q(,)- матрица для маршрутов
80 ' P() - перестановка
90 FOR I=1 TO S:FOR J=1 TO S
100 READ M(I,J)
110 NEXT J,I:CLS
120 LOCATE 8,9:PRINT "проводятся вычисления"
130 FOR I=1 TO S:P(I)=S+1-I:NEXT I
140 Z=1:FOR I=1 TO S-1:Z=Z%I:NEXT I:'-(S-1)!
150 TIME=0:Y=1:GOTO 210:'==========ЯДРО
160 I=2:J=1:'--------------------перестановка
170 IF P(I-1)<=P(I) THEN I=I+1:GOTO 170
180 IF P(J)<=P(I) THEN J=J+1:GOTO 180
190 SWAP P(I),P(J):IF I=2 THEN 210
200 FOR L=1 TO (I-1)/2:SWAP P(L),P(I-L):NEXT
210 A=M(P(S),P(1)):'----------длина маршрута
```

```
220 FOR E=1 TO S-1:A=A+M(P(E),P(E+1)):NEXT
230 IF A>B THEN 260:'---запоминание маршрута
240 IF A<B THEN H=1:B=A
250 FOR E=1 TO S:Q(H,E)=P(E):NEXT:H=H+1
260 Y=Y+1:IF Y<=Z THEN 160
270 CLS:PRINT P$;'====================ВЫВОД
290 PRINT:FOR I=1 TO H-1:PRINT I;")   ";1;
300 FOR J=1 TO S:PRINT Q(I,J);:NEXT:PRINT
310 NEXT I:PRINT
320 PRINT "длина пути  =";B
330 PRINT "время счета =";TIME/50:END
340 'пример 1-------------------------------
350 'DATA 0, 5 ,9, 6 ,3, 5, 9
360 'DATA 8, 0 ,8, 8 ,5, 9, 2
370 'DATA 6, 9 ,0, 1 ,6, 7, 3
380 'DATA 7, 11,4, 0 ,4, 2, 9
390 'DATA 4, 6 ,3, 2 ,0, 2, 8
400 'DATA 5, 2 ,2, 8 ,4, 0, 3
410 'DATA 8, 1 ,3, 16,5, 3, 0
420 'пример 2-------------------------------
430 DATA 0, 3 ,21,6 ,4 ,8, 11
440 DATA 7, 0 ,6 ,2 ,12,5, 3
450 DATA 8, 9 ,0 ,13,8 ,2, 9
460 DATA 6, 10,31,0 ,7 ,7, 14
470 DATA 8, 7 ,6 ,5 ,0 ,4, 3
480 DATA 2, 5 ,7 ,9 ,3 ,0, 6
490 DATA 7, 5 ,4 ,7,15 ,4, 0
500 '-----------------имя программы "КОММИ"
```

Пример 15. Решить «задачу коммивояжера» для 7 городов с именами 1, 2, ..., 7 и матрицей расстояний:

$$M=\begin{pmatrix} 0, & 3, & 21, & 6, & 4, & 8, & 11 \\ 7, & 0, & 6, & 2, & 12, & 5, & 3 \\ 8, & 9, & 0, & 13, & 8, & 2, & 9 \\ 6, & 10, & 31, & 0, & 7, & 7, & 14 \\ 8, & 7, & 6, & 5, & 0, & 4, & 3 \\ 2, & 5, & 7, & 9, & 3, & 0, & 6 \\ 7, & 5, & 4, & 7, & 15, & 4, & 0 \end{pmatrix}.$$

Длина пути между пунктами i и j равна $M(i, j)$.

В тексте программы КОММИ M должно быть зафиксировано в операторах DATA в строках после 330. В нашем случае это строки 430—490. При запуске программы на счет на экране дисплея получаем такое сообщение:

Оптимальные маршруты коммивояжера:

1) 1 2 4 5 7 3 6 1 длина пути=23
время счета=109.1 с

Найденное решение представлено на рисунке 37.

Возможная неоднозначность решения «задачи коммивояжера» учитывается программой КОММИ, запоминание маршрутов с равными длинами осуществляется в отдельных строках двумерного массива Q. Количество зарезервированных строк в нем при необходимости может быть увеличено.

Пример 16. Решить «задачу коммивояжера» при следующей матрице расстояний: .

$$M=\begin{pmatrix} 0, & 5, & 9, & 6, & 3, & 5, & 9 \\ 8, & 0, & 8, & 8, & 5, & 9, & 2 \\ 6, & 9, & 0, & 1, & 6, & 7, & 3 \\ 7, & 11, & 4, & 0, & 4, & 2, & 9 \\ 4, & 6, & 3, & 2, & 0, & 2, & 8 \\ 5, & 2, & 2, & 8, & 4, & 0, & 3 \\ 8, & 1, & 3, & 16, & 5, & 3, & 0 \end{pmatrix}.$$

В тексте программы M зафиксировано в операторах DATA в строках 350—410. После вычислений на экране дисплея получаем такое сообщение.

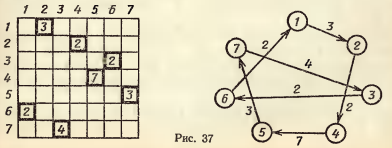


Рис. 37

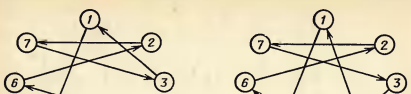


Рис. 38

Оптимальные маршруты коммивояжера:

1)	1	5	4	6	2	7	3	1	длина пути=20
2)	1	5	6	2	7	3	4	1	время счета=108.9 с

Найденные решения представлены на рисунке 38.

2. ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ

2.1. Общая и каноническая задачи

Постановка задачи. Общей задачей линейного программирования (ОЗЛП) называется задача, состоящая в определении оптимального (максимального или минимального) значения линейной функции

$$F=\sum_{j=1}^{n} c_j x_j + c_0 \qquad (1)$$

при условиях

$$\sum_{j=1}^{n} a_{ij} x_j \leftrightarrow b_i \ (i=1, 2, \ldots, m) \qquad (2)$$

$$x_k \geqslant 0 \ (k \in \mathrm{L} \subset \{1, 2, \ldots, n\}) \qquad (3)$$

где: a_{ij}, c_i, b_i — фиксированные действительные числа; $\leftrightarrow$ — один из знаков отношения: $\leqslant$, $\geqslant$, $=$.

Другими словами, в ОЗЛП ищется оптимальное значение линейной функции (1) на множестве решений системы равенств и неравенств (2) при условии неотрицательности некоторых переменных. Заметим, что требование неотрицательности (3) может быть наложено на все переменные ($\mathrm{L}=\{1, 2, \ldots, n\}$) или вообще отсутствовать ($\mathrm{L}=\varnothing$).

Функция F называется *линейной формой* или *целевой функцией* задачи (1)—(3), а условия — (2)—(3) — ограничениями.

Совокупность чисел $X=(x_1, x_2, \ldots, x_n)$, удовлетворяющая ограничениям (2)—(3), называется *допустимым решением* или *планом задачи*.

План $X^*=(x_1^*, x_2^*, \ldots, x_n^*)$, при котором целевая функция принимает минимальное или максимальное значение, называется *оптимальным*.

Таким образом, при любом плане X оптимальный план X^* удовлетворяет условию $F(X^*) \leqslant F(X)$ (или $F(X^*) \geqslant F(X)$).

Не всякая задача линейного программирования имеет оптимальный план. Это связано с тем, что множество решений D системы ограничений (2)—(3) может быть пустым или форма F на D не ограничена снизу (сверху).

Пример 1. Для ОЗЛП $\begin{cases} x_1+x_2 \geqslant 3 \\ x_1 \leqslant 1 \\ x_2 \leqslant 1 \\ x_1,\ x_2 \geqslant 0 \end{cases}$ $F=x_1+x_2 \to \max$

$D=\varnothing$, т. е. ограничения противоречивы, и потому оптимальный план отсутствует.

Пример 2. Для ОЗЛП $\begin{cases} x_1+x_2 \leqslant 2 \\ -x_1+x_2 \geqslant 5 \\ x_2 \geqslant 0 \end{cases}$ $F=x_1+2x_2 \to \min$

при любом $C \geqslant 5$ пара $x_1=-C$, $x_2=0$ является планом. F не ограничена снизу и потому оптимальный план отсутствует.

Среди множества задач линейного программирования выделяют класс так называемых канонических, или основных, задач (КЗЛП), в которых ищется минимум функции (1) при ограничениях (2)—(3), где $\leftrightarrow$ есть $=$, $\mathrm{L}=\{1, 2, \ldots, n\}$. Иными словами, в КЗЛП находится минимальное значение формы F на множестве решений системы уравнений при неотрицательности всех переменных задачи.

Несложными преобразованиями нахождение решения ОЗЛП всегда может быть сведено к решению некоторой канонической задачи. Покажем, как это делается.

1. Прежде всего нахождение максимума целевой функции F заменяется нахождением минимума формы $-F$. Делать это можно потому, что $\max F=-\min(-F)$.

2. Всякое ограничение ОЗЛП вида $\sum_{j=1}^{n} a_{sj} x_j \leqslant b_s$ $(s \in \{1, 2, \ldots, m\})$ заменяется условием-равенством с неотрицательностью новой вспомогательной переменной y_s: $\begin{cases} \sum_{j=1}^{n} a_{sj} x_j + y_s = b_s \\ y_s \geqslant 0. \end{cases}$

3. Всякое ограничение ОЗЛП вида: $\sum_{j=1}^{n} a_{sj} x_j \geqslant b_s$ $(s \in \{1, 2, \ldots, m\})$ заменяется условиями: $\begin{cases} \sum_{j=1}^{n} a_{sj} x_j - y_s = b_s \\ y_s \geqslant 0. \end{cases}$

4. Переменные $x_p\,(p \notin L)$, на которые не наложены условия неотрицательности, представляются в (1) и (2) в виде разности неотрицательных величин: $\begin{cases} x_k = w_k - v_k \\ w_k \geqslant 0,\ v_k \geqslant 0. \end{cases}$

После всех этих преобразований нам удобно опять переобозначить w_k на x_k, а другие введенные вспомогательные неизвестные заменить на x_{n+1}, x_{n+2}, ..., x_{n+t}. Здесь t равно сумме количества неравенств в (2) и количества переменных без ограничений. Тогда получим следующую каноническую задачу:

минимизировать некоторую новую линейную форму F_1 при ограничениях (4)

$$\begin{cases} \sum\limits_{j=1}^{n+t} a_{ij}x_j = b_i \quad (i=1,\ 2,\ ...,\ m) & (5) \\ x_j \geqslant 0 \qquad\qquad (j=1,\ 2,\ ...,\ n+t) & (6) \end{cases}$$

Отметим, что каждому плану

$$(x_1^*,\ x_2^*,\ ...,\ x_n^*) \tag{7}$$

задачи (1)—(3) можно поставить в соответствие некоторый план

$$(x_1^*,\ x_2^*,\ ...,\ x_n^*,\ x_{n+1}^*,\ ...,\ x_{n+t}^*) \tag{8}$$

задачи (4)—(6), где числа $x_j^*(j=1, 2, ..., n+t)$ получаются по той же схеме, по которой реализовывался переход от ОЗЛП к КЗЛП. Обратно, имея план (8) для задачи (4)—(6), получаем план (7) задачи (1)—(3), компоненты которого равны или соответствующим компонентам (8), или их некоторым разностям. Последний случай имеет место при наличии в ОЗЛП переменных без ограничений.

Кроме полученного взаимно однозначного соответствия между планами задач ОЗЛП и КЗЛП, нетрудно видеть, что значения целевой функции F на соответствующих планах одинаковы. Таким образом, оптимальному решению задачи (1)—(3) соответствует оптимальное решение задачи (4)—(6). В этом смысле указанные задачи эквивалентны.

Пример 3. Привести к эквивалентному каноническому виду задачу:

$$\begin{cases} x_1 + x_2 + x_3 \leqslant 7 \\ 2x_1 - x_2 + x_3 \geqslant 2 \\ x_1 - x_2 + 5x_3 = 8 \\ x_1 \geqslant 0 \end{cases} \tag{9}$$

$$F = x_1 + x_2 - x_3 \to \max \tag{10}$$

Сначала избавимся от отсутствия условий неотрицательности на переменные x_2 и x_3. Для этого сделаем замены:

$$x_2 \to x_2 - x_4 \qquad x_3 \to x_3 - x_5,$$

где уже считаем $x_i \geqslant 0\,(i=2, 3, 4, 5)$. Тогда задача (9)—(10) будет выглядеть так:

$$\begin{cases} x_1 + x_2 + x_3 - x_4 - x_5 \leqslant 7 \\ 2x_1 - x_2 + x_3 + x_4 - x_5 \geqslant 2 \\ x_1 - x_2 + 5x_3 + x_4 - 5x_5 = 8 \\ x_i \geqslant 0\ (i=1,\ 2,\ ...,\ 5) \end{cases}$$

$$F = -x_1 - x_2 + x_3 + x_4 - x_5 \to \min$$

Вводя в систему еще две неотрицательные переменные x_6 и x_7, избавимся от неравенств и получаем следующую каноническую задачу:

$$\begin{cases} x_1 + x_2 + x_3 - x_4 - x_5 + x_6 = 7 \\ 2x_1 - x_2 + x_3 + x_4 - x_5 - x_7 = 2 \\ x_1 - x_2 + 5x_3 + x_4 - 5x_5 = 8 \\ x_i \geqslant 0\ (i=1,\ 2,\ ...,\ 7) \end{cases}$$

$$F = -x_1 - x_2 + x_3 + x_4 - x_5 \to \min$$

Симплекс-метод. Для решения задач линейного программирования имеется много различных методов. В особенности это касается конкретных экономических задач. Систематическое исследование ОЗЛП и разработка общих методов их решения были начаты в работах Л. В. Канторовича в 1939 г. В настоящее время основным общим методом решения ОЗЛП является так называемый симплекс-метод, предложенный Данцигом в 1949 году, и многочисленные его модификации. В этом пункте обсудим алгоритм симплекс-метода применительно к решению КЗЛП. Предварительно введем некоторые понятия и обсудим известные факты [20—23].

Множество D допустимых планов задачи (1)—(3) называют многогранником решений ОЗЛП. Может оказаться, что $D=\varnothing$. D представляет собой некоторое подмножество конечномерного Евклидова пространства R_n. Особую роль в многограннике D играют его вершины, т. е. такие точки, которые не могут быть внутренними для любого отрезка, соединяющего две точки D. Обозначим множество вершин D через D_0. Элементы D_0 называются *опорными планами* ОЗЛП. Заметим, что D_0 всегда конечно. Известно, что если $D_0 \neq \varnothing$ и оптимум F существует, то он достигается хотя бы в одной из вершин D_0. В связи с этим утверждением напрашивается такой способ решения ОЗЛП: найти множество D_0 опорных планов и простым перебором определить требуемый оптимум F на D_0. Пусть он равен F_0 и $F(X_0)=F_0$ $(X_0 \in D_0)$. Если при этом удается установить существование искомого оптимума F на D, то ясно, что он равен F_0 и X_0 — соответствующий оптимальный план задачи (1)—(3). Однако практически воспользоваться предложенным способом решения задачи линейного программирования затруднительно. Дело в том, что даже при относительно не большом числе ограничений и неизвестных получается астрономическое количество вершин. Да и сам процесс отыскания вершин не является простой процедурой.

Симплекс-метод решения КЗЛП называют также методом последовательного улучшения плана. А название это объясняется тем, что в отличие от простого перебора вершин и вычисления значений формы F в них в симплекс-методе сначала определяют одну из вершин D, а затем последовательно переходят к соседним (по ребрам) вершинам D_i, в которых значение F ближе к оптимуму.

Итак, рассмотрим каноническую задачу.

$$\begin{cases} \sum_{j=1}^{n} a_{ij}x_j = b_i \quad (i=1, 2, \ldots, m) & (11) \\ x_j \geqslant 0 \qquad\quad (j=1, 2, \ldots, n) & (12) \end{cases}$$

$$b_i \geqslant 0,\ m < n \qquad (13)$$

$$F = \sum_{i=1}^{n} c_i x_i + c_0 \to \min \qquad (14)$$

Условие $b_i \geqslant 0\ (i=1, 2, \ldots, m)$, очевидно, не ограничивает общности. В противном случае соответствующие уравнения можно было бы домножить на -1.

Расширением задачи (11)—(14), или, по-другому, В-задачей, называют следующую КЗЛП:

$$\begin{cases} \sum_{j=1}^{n} a_{ij}\, x_j + x_{n+i} = b_i \quad (i=1, 2, \ldots, m) & (15) \\ x_j \geqslant 0 \qquad\quad (j=1, 2, \ldots, n+m) & (16) \end{cases}$$

$$b_i \geqslant 0,\ m < n \qquad (17)$$

$$F_1 = \sum_{i=1}^{n} c_i x_j + c_0 + B \sum_{j=n+1}^{n+m} x_j \to \min. \qquad (18)$$

Здесь B — достаточно большое число.

Имеется довольно тесная связь между решениями расширенной и исходной задач (см. табл. 4) [20].

Таблица 4

В-задача	Исходная задача
1 Есть оптимальное решение $(x_1^*, x_2^*, \ldots, x_n^*, 0, \ldots, 0)$,	Вектор $(x_1^*, x_2^*, \ldots, x_n^*)$
в котором последние m компонент равны нулю	является оптимальным решением
2 Для всех достаточно больших B есть оптимальное решение $(x_1^*, x_2^*, \ldots, x_n^*, \ldots, x_{n+m}^*)$,	Система ограничений противоречива
в котором хотя бы одна из компонент $x_i^*\ (i>n)$ отлична от нуля	
3 Нет решения (форма не ограничена снизу)	Задача неразрешима или система ограничений противоречива

Кроме того, в В-задаче сразу же просматривается один из опорных планов

$$X = (\underbrace{0,\ 0 \ldots 0}_{n\text{-нулей}},\ b_1,\ b_2,\ \ldots,\ b_m).$$

Роль B в (18) интуитивно ясна. Слагаемые с неотрицательными искусственными переменными $x_i\ (i>n)$ существенно увеличивают значение формы F_1. Поэтому, если в B-задаче есть допустимое решение, в котором все $x_i=0$ при $i>n$, то и в оптимальном решении эти переменные также должны обратиться в нуль. Если в этом решении отбросить последние m компонент, тогда получим план задачи (11)—(14).

В форме F_1 расширенной задачи величины $x_i\ (i>n)$ можно исключить, подставив их значения из (15). Сделаем это:

$$F_1 = \sum_{j=1}^{n} c_j x_j + c_0 + B \sum_{i=1}^{m} x_{n+i} = \sum_{i=1}^{n} c_i x_j + c_0 + B \sum_{i=1}^{m} \left(b_i - \sum_{l=1}^{n} a_{il} x_l\right) =$$

$$= \sum_{j=1}^{n} \left(c_j - B \sum_{i=1}^{m} a_{ij}\right) x_j + c_0 + B \sum_{i=1}^{m} b_i.$$

Введем обозначения

$$\gamma_j = c_j - B \sum_{i=1}^{m} a_{ij}\ (j=1, 2, \ldots, n) \qquad \gamma_0 = c_0 + B \sum_{i=1}^{m} b_i$$

Получаем

$$F_1 = \sum_{j=1}^{n} \gamma_j x_j + \gamma_0 \qquad (19)$$

Для расширенной задачи с преобразованной формой F_1 построим так называемую симплекс-таблицу (см. табл. 5), на которой удобно разъяснить суть симплекс-метода решения КЗЛП. В таблице имеется m единичных векторов, располагаемых в столбцах с $n+1$ по

Таблица 5

Симплекс-таблица задачи (15)—(17), (19)

	0	1	2		n	$n+1$	$n+2$		$n+m$	$n+m+1$
		x_1	x_2		x_n	x_{n+1}	x_{n+2}		x_{n+m}	
0		0	0	...	0	0	0		0	
1	b_1	a_{11}	a_{12}	...	a_{1n}	1	0	...	0	$n+1$
2	b_2	a_{21}	a_{22}	...	a_{2n}	0	1		0	$n+2$
...	...	...	...		...	...	...	...	...	...
m	b_m	a_{m1}	a_{m2}	...	a_{mn}	0	0	...	1	$n+m$
$m+1$	$-\gamma_0$	γ_1	γ_2	...	γ_n	0	0		0	

$n+m$. Эти столбцы и соответствующие им переменные называют *базисными*, а множество базисных переменных — *базисом*. Номера базисных переменных фиксируются в соответствующих строках последнего столбца таблицы. В процессе обработки таблицы по симплекс-методу базис будет меняться. При этом в нулевой строке в столбцах от $n+1$ до $n+m$ будут ставиться метки, фиксирующие факт удаления соответствующей искусственной переменной $x_j\,(j>n)$ из базиса.

В дальнейшем на таблицу 5 нам удобно смотреть как на матрицу A с $m+2$ строками и $n+m+2$ столбцами и обозначать ее текущий элемент через

$$a_{ij}\,(i=0,\ m+1;\ j=0,\ n+m+1)$$

Алгоритм симплекс-метода.

1. Для КЗЛП (11) — (14) строим расширенную задачу и фиксируем симплекс-таблицу $A=(a_{ij})\,(i=0,\ m+1;\ j=0,\ n+m+1)$.
2. **Поиск разрешающего столбца s.**
   В последней строке начиная с $a_{m+1,1}$ и далее находим первый положительный элемент $a_{m+1,s}$, такой, что $a_{0s}=0$. Если подобных s нет, то переходим к пункту 5.
3. **Поиск разрешающей строки k.**

   Пусть для положительных элементов a_{is} столбца s $\max\limits_{1\leqslant i\leqslant m}\dfrac{a_{i0}}{a_{is}}=\gamma$

   и первый из индексов, для которых это отношение достигается, есть k. Если в столбце s в строках от 1 до m положительных элементов нет, то переходим к пункту 6.
4. **Шаг жордановых преобразований.**
   а) Элементы строки k делим на разрешающий элемент a_{ks}:

   $$a_{kj}=a_{kj}\ /\ a_{ks}\,(j=0,\ 1,\ \ldots,\ n+m).$$

   б) Элементы строк и столбцов, отличных от разрешающих, преобразуем по формулам: $a_{ij}=a_{ij}-a_{is}*a_{kj}$

   $(i=1,\ 2,\ \ldots,\ m;\ i\neq k;\ j=0,\ 1,\ 2,\ \ldots,\ n+m;\ j\neq s)$.

   в) Элементы столбца s, кроме a_{ks}, полагаем равными нулю:

   $$a_{is}=0\,(i=1,\ 2,\ \ldots,\ m;\ i\neq k).$$

   г) Пункты а, б, в по сути означают ввод в базис переменной x_s. Отмечаем это в столбце $n+m+1$:

   $$t=a_{k,\ n+m+1};\ a_{k,\ n+m+1}=s.$$

   При этом x_s замещает x_t, и если $t>n$, то этот факт фиксируем меткой: $a_{0t}=1$. Это позволит избежать повторного включения в базис искусственной переменной $x_t\,(t>n)$.
5. Если среди базисных неизвестных есть искусственные $x_t\,(t>n)$, то система ограничений исходной задачи противоречива. В противном случае оптимальный план: $x_j^*=a_{i0}$ $\left(\text{где } \begin{matrix} j=a_{i,\ n+m+1} \\ i=1,\ 2,\ \ldots,\ m \end{matrix}\right)$.

Остальные компоненты оптимального плана равны нулю. При этом

$$\min F=a_{m+1,0}.$$

6. Если среди базисных неизвестных есть искусственные $x_t\,(t>n)$, то система ограничений исходной задачи противоречива. В противном случае целевая функция не ограничена снизу.

Каждая итерация по симплекс-методу приводит, вообще говоря, к уменьшению значения формы F. Однако это значение может сохраниться и прежним. Более того, бывают и такие случаи, когда целевая функция F не изменяется в течение нескольких итераций. Наконец, возможен также возврат и к ранее уже встречавшемуся базису. Последнее явление называется зацикливанием. При решении практических задач зацикливание весьма маловероятно, и потому мы на нем останавливаться не будем. Отметим лишь, что имеются простые способы предупреждения зацикливания.

Уменьшение количества ограничений. При решении конкретных экономических ОЗЛП система ограничений довольно часто содержит условия, относящиеся к отдельным переменным, а не к их совокупностям. Речь идет о неравенствах вида:

$$\alpha\leqslant x\ (\alpha\neq 0)\qquad x\leqslant\beta\qquad \alpha\leqslant x\leqslant\beta. \tag{20}$$

Наличие подобных ограничений при решении задач на ЭВМ заставляет придумывать различного рода неестественные формы ввода данных. Делается это для того, чтобы избежать непосредственного задания большого количества нулевых коэффициентов при неизвестных. Кроме того, условия (20) увеличивают размеры обрабатываемой симплекс-таблицы и, следовательно, время вычислений.

Однако простыми подстановками от большей части неравенства (20) вообще можно избавиться. Пусть множество $\{1,\ 2,\ \ldots,\ n\}$ разбито на 5 непересекающихся подмножеств $t_p\,(p=1,\ 2,\ \ldots,\ 5)$. Рассмотрим постановку ОЗЛП, где условия (20) выделены из общей совокупности ограничений.

$$\begin{cases} \sum\limits_{i=1}^{n} a_{ij}\,x_j \Leftrightarrow b_i & \left(\begin{matrix} i=1,\ 2,\ \ldots,\ m \\ 0<m<n \end{matrix}\right) \\ x_j \text{ без ограничений } & (j\in t_1) \\ \alpha_j\leqslant x_j\ (\alpha_j\neq 0) & (j\in t_2) \\ x_j\leqslant\beta_j & (j\in t_3) \\ \alpha_j\leqslant x_j\leqslant\beta_j & (j\in t_4) \\ 0\leqslant x_j & (j\in t_5) \end{cases} \tag{21}$$

$$F=\sum_{i=1}^{n} c_i x_i+c_0\to \text{opt (оптимум)}. \tag{22}$$

Здесь любые t_p могут оказаться пустыми. Покажем, как перейти к ОЗЛП с условиями неотрицательности на все переменные.

1. Избавляемся от отсутствия ограничений на x_l ($j \in t_1$) так, как это рекомендовано в первом пункте.

2. Последовательно организуем замены переменных:

$$x_j = x_l - \alpha_j \ (j \in t_2) \qquad x_j = \beta_j - x_l \ (j \in t_3).$$

Формы для пересчета коэффициентов в (21) и (22) получаются просто, и мы их не выписываем.

3. Добавляем к (21) неравенства вида: $x_l \leqslant \beta_j$ ($j \in t_4$)

и далее во вновь полученной системе (21) и форме (22) последовательно организуем замены переменных: $x_j = x_l - \alpha_j$ ($j \in t_4$).

В результате всех этих действий получим ОЗЛП с $m + |t_4|$ (через $|t|$ обозначено количество элементов множества t) ограничениями и $n + |t_4|$ неизвестными, каждое из которых неотрицательно.

Программа «Симплекс-1». Приведенная ниже программа «Симплекс-1» решает ОЗЛП (21)—(22) и состоит из 3 частей:

A. Ввод данных о конкретной ОЗЛП и построение симплекс-таблицы соответствующей эквивалентной расширенной канонической задачи с неотрицательными переменными (строки 10—920).

B. Симплексные преобразования (строки 940—1190).

C. Формирование и вывод результатов счета (строки 1200—1370).

Части B и C оформлены в виде единой подпрограммы.

Работа пользователя с программой проводится в интерактивном режиме. В соответствии с запросами на экране дисплея с пульта заносятся данные о ОЗЛП в такой последовательности.

1. Количество уравнений (более 1 переменной).
2. Количество неравенств (более 1 переменной)
3. Общее количество переменных.
4. Количество переменных без ограничений.
5. Количество ограничений вида $B \leqslant x$ ($B \neq 0$).
6. Количество ограничений вида $x \leqslant C$.
7. Количество ограничений вида $B \leqslant x \leqslant C$.
8. Тип решаемой задачи (1 — min, 2 — max).
9. Информация об ограничениях (равенствах и неравенствах) в виде: коэффициенты при x_1, x_2, ..., x_n, знак, свободный член.
10. Информация о линейной форме в виде: коэффициенты при x_1, x_2, ..., x_n, свободный член.
11. Номера переменных без ограничений.
12. Для ограничений вида $B \leqslant x$ ($B \neq 0$): номер переменной, B.
13. Для ограничений вида $x \leqslant C$: номер переменной, C.
14. Для ограничений вида $B \leqslant x \leqslant C$: номер переменной, B, C.

В зависимости от содержания ответов пользователя на обязательные запросы 1—8 дальнейшее направление диалога выбирается программой так, чтобы по возможности упростить ввод данных.

После построения симплекс-таблицы и ее обработки фрагмент C программы «Симплекс-1» формирует и выводит на дисплей одно из трех возможных сообщений:

1. Оптимальное решение:

$$x_1 = A_1$$
$$x_2 = A_2$$
$$\ldots$$
$$x_n = A_n$$
$$\text{MAX (MIN) формы} = B$$

Здесь A_1, A_2, ..., A_n, B — действительные числа.

2. Форма не ограничена сверху (снизу).
3. Система ограничений противоречива.

Пример 4. Программа «Симплекс-1»

```
10 '          ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ.
20 '          СИМПЛЕКС-МЕТОД  (симплекс 1)
30 '-------------------------------------------------
40  SCREEN 0:COLOR 15,4:WIDTH 40:KEY OFF
50 B=100000000!:EPS=1E-10
60 J1$=" 5.ограничений вида: B<=X    (B <>0)"
70 J2$="ввод коэффициентов линейной формы:"
80 J3$=" 1.уравнений  (более 1 переменной)"
90 J4$=" 2.неравенств (более 1 переменной)"
100 J5$="система ограничений противоречива"
110 J6$="форма неограничена снизу"
120 J7$="форма неограничена сверху"
130 J8$="MIN формы = ":J9$="MAX формы = "
140 I1$="введите через запятые количество:"
150 PRINT "С И М П Л Е К С - М Е Т О Д"
160 '----------ФОРМИРОВАНИЕ СИМПЛЕКС-ТАБЛИЦЫ
170 PRINT I1$:PRINT:PRINT J3$:PRINT J4$
180 PRINT " 3.переменных"
190 PRINT " 4.переменных без ограничений"
200 PRINT J1$
210 PRINT " 6.ограничений вида:    X<=C"
```

```
220 PRINT " 7.ограничений вида: B<=X<=C"
230 PRINT " 8.тип задачи (min - 1, max - 2)"
240 PRINT:INPUT M1,M2,N1,N2,N3,N4,N5,T
250 M=M1+M2+N5:M9=M1+M2+1:T=3-2*T
260 N=N1+N2+M2+N5:N7=N1+1
270 NB=N7+N2:N9=N+1:M0=M+1:N0=N+M0
280 DIM A(M0,N0),A$(M)
290 DIM X(N1),B(N1),C(N1),Y(N1+N2):PRINT
300 '----------------------система ограничений
310 PRINT "ввод коэффициентов ограничений:"
320 IF M1+M2=0 THEN 420 ELSE PRINT
330 FOR I=1 TO M1+M2:FOR J=1 TO N7
340 R=J:IF J<>N7 THEN 390 ELSE R=0
350 INPUT "знак           ";A$(I)
360 IF A$(I)="=" THEN 390
370 IF A$(I)="<=" THEN 390
380 IF A$(I)=">=" THEN 390 ELSE 350
390 PRINT "A(";I;",";J;") = ";:INPUT A(I,R)
400 NEXT J:PRINT :NEXT I
410 '-------------------------линейная форма
420 PRINT J2$:PRINT
430 FOR J=1 TO N7
440 PRINT "F(";J;") = ";:INPUT A(M0,J)
450 NEXT J:PRINT
460 A(M0,0)=-A(M0,N7):A(M0,N7)=0
470 '----------------------X без ограничений
480 IF N2=0 THEN 550
490 PRINT "для X без ограничений:":PRINT
500 L=N7
510 FOR K=1 TO N2:INPUT "номер X";J:X(J)=2
```



```
520 FOR I=1 TO M0:A(I,L)=-A(I,J):NEXT I
530 L=L+1:NEXT K:PRINT
540 '-----------------------------B<=X (X<>0)
550 IF N3=0 THEN 630
560 PRINT "для X с условием: B<=X (B<>0)
570 FOR K=1 TO N3
580 INPUT "номер X,B";J,R:X(J)=3:B(J)=R
590 FOR I=1 TO M0
600 A(I,0)=A(I,0)-B(J)*A(I,J)
610 NEXT I,K:PRINT
620 '-----------------------------------X<=C
630 IF N4=0 THEN 710
640 PRINT "для X с условием: X<=C"
650 FOR K=1 TO N4
660 INPUT "номер X,C";J,C:X(J)=4:C(J)=C
670 FOR I=1 TO M0
680 A(I,J)=-A(I,J):A(I,0)=A(I,0)+C(J)*A(I,J)
690 NEXT I,K
700 '-----------------------------------B<=X<=C
710 IF N5=0 THEN 800
720 PRINT "для X с условием: B<=X<=C:L=M9
730 FOR K=1 TO N5:INPUT "номер X,B,C";J,R,C
740 X(J)=5:B(J)=R:C(J)=C:A(L,J)=1
750 A(L,0)=C(J):A$(L)="<=":L=L+1
760 FOR I=1 TO M0
770 A(I,0)=A(I,0)-B(J)*A(I,J)
780 NEXT I,K:PRINT
790 '----неравенства в =,свободные члены >=0
800 L=N8:FOR I=1 TO M
810 IF A$(I)="<=" THEN A(I,L)=1:L=L+1
```

```
820 IF A$(I)=">=" THEN A(I,L)=-1:L=L+1
830 IF A(I,0)>=0 THEN 850
840 FOR S=0 TO N:A(I,S)=-A(I,S):NEXT S
850 NEXT I
860 ' ---E - матрица,# базисных X,пересчет F
870 FOR I=1 TO M:A(I,N+I)=1:A(I,N0)=N+I:NEXT
880 FOR J=0 TO N:S=0
890 FOR I=1 TO M:S=S+A(I,J):NEXT I
900 A(M0,J)=B%S-A(M0,J)%T
910 NEXT J:P=N+M:CLS
920 LOCATE 8,9:PRINT "проводятся вычисления"
930 TIME=0:GOSUB 940:END
940 '=====================================ЯДРО
950 '-----------поиск разрешающего столбца S
960 S=0:FOR J=1 TO P
970 IF A(M0,J)>EPS AND A(0,J)<>1 THEN S=J:J=P
980 NEXT J:IF S<>0 THEN 1010
990 CLS:IF P>N THEN 1350 ELSE 1210
1000 '-----------поиск разрешающей строки K
1010 R=B:K=0:FOR I=1 TO M
1020 IF A(I,S)<=EPS THEN 1040
1030 C=A(I,0)/A(I,S):IF C<R THEN R=C:K=I
1040 NEXT I:IF K<>0 THEN 1070
1050 CLS:IF P>N THEN 1350 ELSE 1360
1060 '-------------шаг жордановых исключений
1070 R=A(K,S)
1080 FOR J=0 TO P:A(K,J)=A(K,J)/R:NEXT J
1090 FOR I=1 TO M0:IF I=K THEN 1130
1100 FOR J=0 TO P
1110 IF J<>STHEN A(I,J)=A(I,J)-A(I,S)%A(K,J)
```



```
1120 NEXT J
1130 NEXT I
1140 FOR I=1 TO M0
1150 IF I<>K THEN A(I,S)=0
1160 NEXT I:R=A(K,N0):A(K,N0)=S
1170 IF R>N THEN A(0,R)=1:W=W+1
1180 '---------------- усечение столбцов до N
1190 IF W<P-N THEN 960 ELSE P=N:GOTO 960
1200 '=====================================ВЫВОД
1210 PRINT "оптимальное решение:":PRINT
1220 FOR J=1 TO N1+N2:C=0
1230 FOR K=1 TO M:IF A(K,N0)=J THEN C=A(K,0)
1240 NEXT K:Y(J)=C:NEXT J
1250 L=N7:FOR J=1 TO N1:
1260 IF X(J)=2 THEN:Y(J)=Y(J)-Y(L):L=L+1
1270 IF X(J)=3 OR X(J)=5 THEN Y(J)=Y(J)+B(J)
1280 IF X(J)=4 THEN:Y(J)=C(J)-Y(J)
1290 PRINT "X(";J;") =";
1300 PRINTUSING "######.###";Y(J)
1310 NEXT J:PRINT
1320 IF T=1 THEN PRINT J8$; ELSE PRINT J9$;
1330 PRINTUSING "########.###";A(M0,0)%T
1340 GOTO 1370
1350 PRINT J5$:GOTO 1370
1360 IF T=1 THEN PRINT J6$ ELSE PRINT J7$
1370 PRINT "время = ";TIME/50:RETURN
1380 '====имя программы на диске-СИМПЛЕКС.INP
```

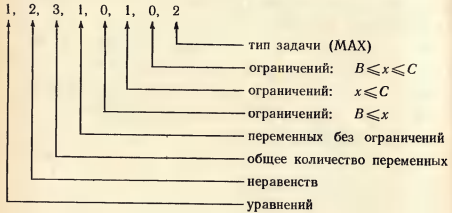
Приведем несколько примеров работы с программой «Симплекс-1».

Пример 5. Решить задачу

$$\begin{cases} x_1 + x_2 - 3x_3 \geqslant 7 \\ x_1 + 3x_2 + x_3 \leqslant 15 \\ x_1 - x_2 + x_3 = 3 \\ x_3 \leqslant 4,\ x_2 \geqslant 0 \end{cases}$$

$$F = x_1 + x_2 + x_3 \to \max$$

Первая строка вводимых данных



Остальные данные.

1, 1, −3, ⩾, 7
1, 3, 1, ⩽, 15
1, −1, 1, =, 3
← информация об ограничениях

1, 1, 1, 0 ← информация о форме F

1 ← номер переменной без ограничения

3, 4 ← информация об ограничениях вида: $X \leqslant C$

В результате счета получаем сообщение:

ОПТИМАЛЬНОЕ РЕШЕНИЕ
$x_1 = 5.5$
$x_2 = 3$
$x_3 = 0.5$
MAX ФОРМЫ = 9

Пример 6. Решить задачу

$$\begin{cases} x_1 + x_2 \leqslant 1 \\ x_1 - x_2 \geqslant 0 \\ x_2 \geqslant 0.5 \end{cases}$$

$$F = 3x_1 - x_2 \to \min$$

Вводимые данные:
0, 2, 2, 1, 1, 0, 0, 1
1, 1, ⩽, 1
1, −1, ⩾, 0
3, −1, 0
1
2, 0.5

Сообщение:

СИСТЕМА ОГРАНИЧЕНИЙ ПРОТИВОРЕЧИВА

Пример 7. Решить задачу

$$\begin{cases} x_1 + x_2 \leqslant 5 \\ x_1 - x_2 \leqslant 1 \\ x_1 - 5x_2 \geqslant -10 \end{cases}$$

$$F = x_1 + x_2 + 1 \to \min$$

Вводимые данные:
0, 3, 2, 2, 0, 0, 0, 1
1, 1, ⩽, 5
1, −1, ⩽, 1
1, −5, ⩾, −10
1, 1, 1
1, 2

Сообщение:

ФОРМА НЕ ОГРАНИЧЕНА СНИЗУ

Программа «Симплекс-2». Довольно простая организация ввода данных в программе «Симплекс-1» требует совершенствования. Сбой при вводе конкретного значения не должен приводить к повторному вводу всех данных задачи. Необходимо также иметь возможность редактировать уже введенные данные. Простейший способ добиться этого состоит в полном отказе от использования оператора INPUT. Данные можно заготавливать в подпрограмме в операторах DATA и считывать их по мере надобности. В этом случае в распоряжении пользователя оказывается все многообразие сервисных средств корректирования данных, предусмотренных экранным редактором текстов. Именно так и решаются ОЗЛП (21)—(22) по программе «Симплекс-2». Причем данные в DATA необходимо располагать в той же последовательности, как они вводились бы по программе «Симплекс-1». Обработка и сообщения о результатах счета в обеих программах реализуются фрагментом, начинающимся со строки 940 и далее.

Пример 8. Программа «Симплекс-2»

```
10 '          ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ
20 '       СИМПЛЕКС МЕТОД  (симплекс 2)
30 '=========================================
```

```
40 'ВНИМАНИЕ !!!          И Н С Т Р У К Ц И Я
50 '  для пользователя в строках 70-280
60 '-----------------------------------------
70 'расположить данные в DATA со строки 1400
80 ' M1 - количество уравнений
90 ' M2 - количество неравенств
100 'N1 - общее количество переменных
110 'N2 - кол. переменных без ограничений
120 'N3 - кол. ограничений вида: B<=X (B<>0)
130 'N4 - кол. ограничений вида: X<=C
140 'N5 - кол. ограничений вида: B<=X<=C
150 'T  -  тип задачи (min - 1, max - 2)
160 ' Для всех граничений между переменными:
170 ' коэффициенты при X(1),X(2),... X(N1),
180 ' знак,свободный член.
190 ' Для линейной формы:
200 ' коэффициенты при  X(1),X(2),...X(N1),
210 ' свободный член.
220 ' Номер X -для всех переменных X без
230 ' ограничений
240 ' Номер X,B   -для ограничений вида:
250 ' B<=X        (B не нуль)
260 ' Номер X,C  -для ограничений вида: X<=C
270 ' Номер X,B,C -для ограничений вида:
280 ' B<=X<=C
290 '==========ФОРМИРОВАНИЕ СИМПЛЕКС-ТАБЛИЦЫ
300 SCREEN 0:COLOR 15,4:KEY OFF
310 J1$="MIN форма = ":J2$="MAX форма = "
320 J3$="система ограничений противоречива"
330 J4$="форма неограничена снизу"
```



```
340 J5$="форма неограничена сверху"
350 LOCATE 8,9:PRINT "проводятся вычисления"
360 B=1000000000!:EPS=1E-10:'---чтение данных
370 READ M1,M2,N1,N2,N3,N4,N5,T
380 M=M1+M2+N5:M9=M1+M2+1:T=3-2*T
390 N=N1+N2+M2+N5
400 N7=N+1:N8=N7+N2:N9=N+1:M0=M+1:N0=N+M0
410 DIM A(M0,N0),A$(M)
420 DIM X(N1),B(N1),C(N1),Y(N1+N2)
430 FOR I=1 TO M9:FOR J=1 TO N7
440 IF J<=N1 AND I<M9 THEN READ A(I,J)
450 IF J>N1 AND I<M9 THEN READ A$(I),A(I,0)
460 IF I=M9 THEN READ A(M0,J)
470 NEXT J,I:A(M0,0)=-A(M0,N7):A(M0,N7)=0
480 IF N2=0 THEN 530
490 '---------обработка X(J) без ограничений
500 L=N7:FOR K=1 TO N2:READ J:X(J)=2
510 FOR I=1 TO M0:A(I,L)=-A(I,J):NEXT I
520 L=L+1:NEXT K
530 IF N3=0 THEN 590
540 '-----обработка X(J) с условием: B<=X(J)
550 FOR K=1 TO N3:READ J:X(J)=3:READ B(J)
560 FOR I=1 TO M0
570 A(I,0)=A(I,0)-B(J)*A(I,J)
580 NEXT I,K
590 IF N4=0 THEN 650
600 '-----обработка X(J) с условием: X(J)<=C
610 FOR K=1 TO N4:READ J:X(J)=4:READ C(J)
620 FOR I=1 TO M+1
630 A(I,J)=-A(I,J):A(I,0)=A(I,0)+C(J)*A(I,J)
```

```
640 NEXT I,K
650 IF N5=0 THEN 740
660 '--обработка X(J) с условием: B<=X(J)<=C
670 L=M9: FOR K=1 TO N5
680 READ J:X(J)=5:READ B(J),C(J)
690 A(L,J)=1:A(L,0)=C(J):A$(L)="<=":L=L+1
700 FOR I=1 TO M0
710 A(I,0)=A(I,0)-B(J)*A(I,J)
720 NEXT I,K
730 '----неравенства в =,свободные члены >=0
740 L=N8:FOR I=1 TO M
750 IF A$(I)="<=" THEN A(I,L)=1:L=L+1
760 IF A$(I)=">=" THEN A(I,L)=-1:L=L+1
770 IF A(I,0)>=0 THEN 790
780 FOR S=0 TO N:A(I,S)=-A(I,S):NEXT S
790 NEXT I
800 '------вспомогательный базис, пересчет F
810 L=N9:FOR I=1 TO M:Q=0
820 FOR J=1 TO N:IF A(I,J)<>1 THEN 860
830 R=0:FOR X=1 TO M
840 IF X<>I AND A(X,J)<>0 THEN R=1:X=M
850 NEXT X:IF R=0 THEN P=J:J=N:Q=1
860 NEXT J
870 IFQ=0THEN A(I,L)=1:A(M0,L)=B*T:P=L:L=L+1
880 A(I,N0)=P:NEXT I:P=L-1
890 FOR J=0 TO P:S=0
900 FOR I=1 TO M:S=S+A(I,J):NEXT I
910 A(M0,J)=B*S-A(M0,J)*T
920 NEXT J:TIME=0:GOSUB 930:END
930 '==================================ЯДРО
```



```
940 '----------поиск разрешающего столбца S
950 S=0:FOR J=1 TO P
960 IF A(M0,J)>EPS AND A(0,J)<>1 THENS=J:J=P
970 NEXT J:IF S<>0 THEN 1000
980 CLS:IF P>N THEN 1310 ELSE 1180
990 '----------поиск разрешающей строки K
1000 R=B:K=0:FOR I=1 TO M
1010 IF A(I,S)<=EPS THEN 1030
1020 C=A(I,0)/A(I,S):IF C<R THEN R=C:K=I
1030 NEXT I:IF K<>0 THEN 1060
1040 CLS:IF P>N THEN 1310 ELSE 1320
1050 '----------шаг жордановых исключений
1060 R=A(K,S)
1070 FOR J=0 TO P:A(K,J)=A(K,J)/R:NEXT J
1080 FOR I=1 TO M0:IF I=K THEN 1120
1090 FOR J=0 TO P
1100 IF J<>STHEN A(I,J)=A(I,J)-A(I,S)*A(K,J)
1110 NEXT J
1120 NEXT I
1130 FOR I=1 TO M0:IF I<>K THEN A(I,S)=0
1140 NEXT I:R=A(K,N0):A(K,N0)=S
1145 IF R>N THEN A(0,R)=1:W=W+1
1150 IF W<P-N THEN 950
1160 P=N:GOTO 950:'---усечение столбцов до N
1170 '==================================ВЫВОД
1180 PRINT "оптимальное решение:":PRINT
1190 FOR J=1 TO N1+N2:C=0
1200 FOR K=1 TO M:IF A(K,N0)=J THEN C=A(K,0)
1210 NEXT K:Y(J)=C:NEXT J
1220 L=N7:FOR J=1 TO N1:
```

```
1230 IF X(J)=2 THEN:Y(J)=Y(J)-Y(L):L=L+1
1240 IF X(J)=3 OR X(J)=5 THEN Y(J)=Y(J)+B(J)
1250 IF X(J)=4 THEN:Y(J)=C(J)-Y(J)
1260 PRINT "X(";J;") =";
1270 PRINTUSING "######.###";Y(J)
1280 NEXT J:PRINT
1290 IF T=1 THEN PRINT J1$; ELSE PRINT J2$;
1300 PRINTUSING "########.###";A(M0,0)*T
1305 GOTO 1330
1310 PRINT J3$:GOTO 1330
1320 IF T=1 THEN PRINT J4$ ELSE PRINT J5$
1330 PRINT "время = ";TIME/50:RETURN
1340 '===имя программы на диске-СИМПЛЕКС.DAT
```

Примеры решения задач с помощью программы «Симплекс-2».

Пример 9. Для решения ОЗЛП из примера 4 достаточно сформировать фрагмент:

```
2000 DATA 1, 2, 3, 1, 0, 1, 0, 2
2010 DATA 1, 1, -3, >=, 7
2020 DATA 1, 3, 1, <=, 15
2030 DATA 1, -1, 1, =, 3
2040 DATA 1, 1, 1, 0
2050 DATA 1
2060 DATA 3, 4
```

(23)

Сообщение о результатах счета, естественно, будет таким же, как и в примере 4.

Заметим, что поскольку блок данных создается интерпретатором в оперативной памяти сразу же после запуска программы, то специального обращения к (23) не требуется.

Пример 10. Решить задачу

$$\begin{cases} x_1 + x_4 - 2x_5 = 1 \\ x_2 - 2x_4 + x_5 = 2 \\ x_3 + 3x_4 + x_5 = 3 \\ x_i \geqslant 0 \quad (i = \overline{1,5}) \end{cases}$$

$$F = x_4 - x_5 \to \min$$

Фрагмент с данными запишем так.

```
2000 DATA 3, 0, 5, 0, 0, 0, 0, 1
2010 DATA 1, 0, 0, 1, -2,=,1
2020 DATA 0, 1, 0, -2, 1,=, 2
2030 DATA 0, 0, 1, 3, 1,=, 3
2040 DATA 0, 0, 0, 1, -1, 0
```

Сообщение:

ОПТИМАЛЬНОЕ РЕШЕНИЕ

$x_1 = 5.6$
$x_2 = 0$
$x_3 = 0$
$x_4 = 0.2$
$x_5 = 2.4$

MIN формы $= -2.2$

В заключение этого пункта добавим, что программа «Симплекс-2» работает, вообще говоря, быстрее программы «Симплекс-1». Дело в том, что по ней более экономно создаются симплекс-таблицы. Скажем, в последнем примере переменные x_1, x_2 и x_3 уже составляют базис задачи. Этот факт учитывается программой «Симплекс-2», и дополнительные искусственные переменные вводиться не будут. В общем случае при построении базиса количество вводимых искусственных переменных минимизируется. Подобный анализ программа «Симплекс-1» не проводит.

2.2. Транспортная задача

Постановка задачи. Одной из частных задач линейного программирования является транспортная задача, состоящая в наиболее рациональном закреплении пунктов отправления некоторого однородного продукта к пунктам назначения. При этом в качестве критерия оптимальности обычно берется либо минимальная стоимость перевозок требуемого количества груза, либо минимальное время его доставки. Важность этой задачи для приложений и специфика получающихся ограничений позволили разработать более эффективные по сравнению с симплекс-методом алгоритмы ее решения. К ним можно отнести распределительный метод, метод дифференциальной ренты, метод потенциалов, венгерский алгоритм и т. д. Существуют матричная и сетевая постановки транспортной задачи.

Мы будем заниматься первой из них, где в качестве критерия оптимальности берется минимизация стоимости всех перевозок.

Перейдем к более точной формулировке задачи.

В пунктах отправления A_i сосредоточены соответственно запасы a_i $(i = \overline{1, m})$ единиц однородного груза. Этот груз следует доставить

в пункты B_j в количествах b_j ($j=\overline{1,\ n}$). Выполнено условие баланса, т. е. совокупный запас груза в A_i ($i=\overline{1,\ m}$) равен общей его потребности в B_j ($j=\overline{1,\ n}$). Кроме того, известны $m\times n$ тарифов (издержек, стоимостей, транспортных расходов): C_{ij} ($i=\overline{1,\ m}$; $j=\overline{1,\ n}$) на перевозку единицы груза из A_i в B_j. Требуется организовать перевозки за минимум стоимости.

Планом задачи назовем всякое ее решение из $m\times n$ чисел x_{ij} ($i=\overline{1,\ m}$; $j=\overline{1,\ n}$), представляющих собой объемы перевозок из A_i в B_j.

Матрицы $C=(c_{ij})$ и $X=(x_{ij})$ ($i=\overline{1,\ m}$; $j=\overline{1,\ n}$) будем называть соответственно матрицами стоимостей и транспортных перевозок.

Теперь мы в состоянии написать математическую модель транспортной задачи (ТЗ).

Найти матрицу перевозок $X=(x_{ij})$, доставляющую минимум форме

$$F=\sum_{j=1}^{n}\sum_{i=1}^{m} c_{ij}\,x_{ij} \tag{1}$$

при следующих ограничениях.

1. Все грузы должны быть вывезены:

$$\sum_{j=1}^{n} x_{ij}=a_i\ (i=\overline{1,\ m};\ a_i\geqslant 0). \tag{2}$$

2. Все потребности должны быть удовлетворены:

$$\sum_{i=1}^{m} x_{ij}=b_j\ (j=\overline{1,\ n};\ b_j\geqslant 0). \tag{3}$$

3. Движение грузов должно происходить от пунктов отправления к пунктам назначения:

$$x_{ij}\geqslant 0\ (i=\overline{1,\ m};\ j=\overline{1,\ n}). \tag{4}$$

4. Выполнено условие сбалансированности, или, по-другому, замкнутости задачи:

$$\sum_{i=1}^{m} a_i=\sum_{j=1}^{n} b_j=M\ (M>0). \tag{5}$$

Нетрудно видеть, что мы получили каноническую задачу линейного программирования. Более того, в ней коэффициенты при всех $m\times n$ переменных x_{ij} равны 1, а каждая переменная входит лишь в два уравнения. Далее, задача (1)—(5) всегда имеет решение. Докажем это.

Прежде всего множество D решений системы ограничений (2)—(4) не пусто. Действительно, величины $x_{ij}=(a_i\cdot b_j)/M$ ($i=\overline{1,\ m}$; $j=\overline{1,\ n}$) удовлетворяют условиям (2)—(4):

$$\sum_{j=1}^{n} x_{ij}=\sum_{j=1}^{n}(a_i\cdot b_j)/M=(a_i/M)\sum_{j=1}^{n} b_j=a_i\ (i=\overline{1,\ m})$$

$$\sum_{i=1}^{m} x_{ij}=\sum_{i=1}^{m}(a_i\cdot b_j)/M=(b_j/M)\sum_{i=1}^{m} a_i=b_j\ (j=\overline{1,\ n})$$

$$x_{ij}\geqslant 0\ (i=\overline{1,\ m};\ j=\overline{1,\ n}).$$

Кроме того, многогранник D есть ограниченное множество Евклидова пространства E_{nm}: $0\leqslant x_{ij}\leqslant \max_{ij}(a_i,\ b_j)$. Эти факты и означают существование решения транспортной задачи (ТЗ).

Отметим, что замкнутость ТЗ, задаваемая соотношением баланса (5), является необходимым и достаточным условием существования решения системы ограничений и ТЗ в целом. Достаточность (5) мы уже фактически установили, а его необходимость доказывается так. Пусть (2)—(4) совместна. Тогда

$$\sum_{i=1}^{m} a_i=\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}=\sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}=\sum_{j=1}^{n} b_j.$$

Наряду с рассмотренной классической замкнутой ТЗ иногда приходится иметь дело с открытыми моделями ТЗ, в которых нарушен баланс запасов и потребностей. Остановимся на случаях.

А. **ТЗ с избытком запасов:**

$$b_{n+1}=\sum_{i=1}^{m} a_i-\sum_{j=1}^{n} b_j>0. \tag{6}$$

Для отыскания оптимального плана в случае (6) вводят фиктивный $(n+1)$-й пункт назначения B_{n+1} с потребностью b_{n+1} и полагают стоимости перевозок грузов в B_{n+1} равными нулю. Полученная новая ТЗ является уже замкнутой. Найдя оптимальный план (x_{ij}) ($i=\overline{1,\ m}$; $j=\overline{1,\ n+1}$) этой ТЗ и отбрасывая в полученной матрице последний столбец, получим оптимальный план исходной ТЗ.

В. **ТЗ с избытком заявок:**

$$a_{m+1}=\sum_{j=1}^{n} b_j-\sum_{i=1}^{m} a_i>0. \tag{7}$$

Эта задача несколько сложнее предыдущей. Дело в том, что, как ни развози грузы, все заявки удовлетворить все равно не удастся. Поэтому постановка задачи нуждается в уточнении.

1. Все пункты назначения требуется удовлетворить пропорционально поданным заявкам.

В этом случае, подсчитав коэффициент пропорциональности

$$k=\Big(\sum_{i=1}^{m} a_i\Big)\Big/\Big(\sum_{j=1}^{n} b_j\Big)$$

и «подправив» заявки: $\beta_j=k\cdot b_j$ ($j=\overline{1,\ n}$), получим замкнутую ТЗ.

2. Если вовсе не заботиться о «справедливости» удовлетворения заявок, а по-прежнему интересоваться лишь минимизацией транспортных расходов, то для отыскания оптимального плана при условии (7) вводят фиктивный $(m+1)$-й пункт отправления A_{m+1} с запасом груза a_{m+1} и полагают стоимости перевозок грузов из A_{m+1} равными нулю. Полученная ТЗ является замкнутой.

Найдя оптимальный план x_{ij} $(i=\overline{1, m+1};\ j=\overline{1, n})$ этой ТЗ и отбрасывая в полученной матрице последнюю строку, получим оптимальный план исходной ТЗ.

В заключение пункта отметим, что многие экономические задачи, далекие по постановке от ТЗ, приводят к той же самой математической модели (1)—(4). Ограничимся одним примером.

Имеется m видов сельскохозяйственных культур и n хозяйств, где их можно выращивать. Из-за различных условий доход, получаемый с 1 га посева одной и той же культуры, в разных хозяйствах неодинаков. Обозначим его для i-й культуры и j-го хозяйства через c_{ij}. Общие площади посева культур a_i $(i=\overline{1, m})$ и площади пашни в хозяйствах b_j $(j=\overline{1, n})$ известны. Требуется составить такой план размещения сельскохозяйственных культур по хозяйствам, чтобы общий доход был максимальным.

Обозначим площадь посева i-й культуры в j-м хозяйстве через x_{ij}. Тогда получаем задачу.

Найти план $X=(x_{ij})$ такой, что: $\sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n} c_{ij}\,x_{ij} \to \max$

при условиях:

а) план посева по каждой культуре должен быть выполнен

$$\sum_{j=1}^{n} x_{ij} = a_i \quad (i=\overline{1, m})$$

б) пашня в хозяйствах должна быть использована полностью

$$\sum_{i=1}^{m} x_{ij} = b_j \quad (j=\overline{1, n})$$

в) $x_{ij} \geqslant 0$.

Мы получили задачу, аналогичную транспортной, но здесь ищется максимум линейной формы.

Метод потенциалов. В этом пункте описывается метод потенциалов решения ТЗ, предложенный Л. В. Канторовичем и М. К. Гавуриным еще до разработки симплекс-метода. Общий принцип определения оптимального плана методом потенциалов аналогичен симплекс-методу, а именно: сначала находят опорный план ТЗ, а затем его последовательно улучшают. Итак: рассмотрим замкнутую ТЗ.

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\,x_{ij} \to \min \tag{8}$$

при условиях:

$$\sum_{j=1}^{n} x_{ij} = a_i \quad (i=\overline{1, m}) \qquad \sum_{i=1}^{m} x_{ij} = b_j \quad (j=\overline{1, n})$$

$$x_{ij} \geqslant 0 \quad (i=\overline{1, m};\ j=\overline{1, n}) \qquad \sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j. \tag{9}$$

Объяснять алгоритм метода потенциалов удобно на специальном макете, который выглядит так, как показано в таблице 6.

Таблица 6

Макет транспортной задачи

	B_1	B_2		B_n	
A_1	$\frac{c_{11}}{x_{11}}$	$\frac{c_{12}}{x_{12}}$	..	$\frac{c_{1n}}{x_{1n}}$	a_1
A_2	$\frac{c_{21}}{x_{21}}$	$\frac{c_{22}}{x_{22}}$	...	$\frac{c_{2n}}{x_{2n}}$	a_2
..	...				...
A_m	$\frac{c_{m1}}{x_{m1}}$	$\frac{c_{m2}}{x_{m2}}$		$\frac{c_{mn}}{x_{mn}}$	a_m
	b_1	b_2	...	b_n	

Остановимся прежде всего на одном из способов построения первоначального опорного плана ТЗ. Речь идет о так называемом методе «северо-западного угла». Суть его состоит в следующем.

Пусть вначале все $x_{ij}=0$ $(i=\overline{1, m};\ j=\overline{1, n})$. Будем последовательно менять величины x_{ij} начиная с клетки (1, 1). На каждом шаге, невзирая на тарифы, назначим перевозку от одного пункта отправления к одному пункту назначения так, чтобы либо исчерпался имеющийся запас груза в первом пункте, либо удовлетворилась потребность в грузе во втором пункте. В любом из этих вариантов один из пунктов исключим из дальнейшего рассмотрения. Пункты A_m и B_n должны быть исключены одновременно.

Более четко процедуру «северо-западного угла» можно понять из примеров, представленных макетами таблиц 7 и 8.

Пример 11. В таблице приведен первоначальный план, построенный способом «северо-западного угла». Стоимости в клетках не проставлены. При построении плана они не используются.

Опишем шаги заполнения таблицы 7.

1. $x_{ij}=0$ $(i=\overline{1, 3};\ j=\overline{1, 4})$

Таблица 7

Построение начального плана ТЗ методом «северо-западного угла»

	B_1	B_2	B_3	B_4	
A_1	10	20	0	0	30
A_2	0	30	10	0	40
A_3	0	0	20	40	60
	10	50	30	40	

Таблица 8

Построение начального плана ТЗ методом «северо-западного угла»

	B_1	B_2	B_3	B_4	
A_1	$\underline{0}$	20	$\underline{0}$	0	20
A_2	0	0	30	$\underline{0}$	30
A_3	0	0	0	50	50
	0	20	30	50	

2. Перевозка из A_1 в B_1. Исключение B_1

$a_1=30>b_1=10$ $\quad x_{11}=b_1,\ a_1=a_1-b_1=20,\ b_1=0.$

3. Перевозка из A_1 в B_2. Исключение A_1

$a_1=20<b_2=50$ $\quad x_{12}=a_1,\ b_2=b_2-a_1=30,\ a_1=0.$

4. Перевозка из A_2 в B_2. Исключение B_2

$a_2=40>b_2=30$ $\quad x_{22}=b_2,\ a_2=a_2-b_2=10,\ b_2=0$

5. Перевозка из A_2 в B_3. Исключение A_2

$a_2=10<b_3=30$ $\quad x_{23}=a_2,\ b_3=b_3-a_2=20,\ a_2=0.$

6. Перевозка из A_3 в B_3. Исключение B_3

$a_3=60>b_3=20$ $\quad x_{33}=b_3,\ a_3=a_3-b_3=40,\ b_3=0.$

7. Перевозка из A_3 в B_4. Исключение A_3 и B_4

$a_3=40=b_3$ $\quad x_{34}=b_3,\ a_3=b_3=0.$

Пример 12. В первоначальном плане таблицы 8 подчеркнутые нули в макете — это те нули, которые вписывались в нее по методу «северо-западного угла».

Здесь шаги заполнения таблицы 8 выглядят так.

1. $x_{ij}=0$ $(i=\overline{1,3};\ j=\overline{1,4})$

2. Перевозка из A_1 в B_1. Исключение B_1

$a_1=20>b_1=0$ $\quad x_{11}=b_1,\ a_1=a_1-b_1=20,\ b_1=0.$

В макете x_{11} отмечен нулем с подчеркиванием.

3. Перевозка из A_1 в B_2. Исключение B_2

$a_1=20=b_1$ $\quad x_{12}=b_1,\ a_1=a_1-b_1=0,\ b_1=0.$

4. Перевозка из A_1 в B_3. Исключение A_1

$a_1=0<b_3=30$ $\quad x_{13}=a_1,\ b_3=b_3-a_1=30,\ a_1=0.$

5. Перевозка из A_2 в B_3. Исключение B_3

$a_2=30=b_3$ $\quad x_{23}=b_3,\ a_2=a_2-b_3=0,\ b_3=0.$

6. Перевозка из A_2 в B_4. Исключение A_2

$a_2=0<b_4=50$ $\quad x_{24}=a_2,\ b_4=b_4-a_2=50,\ a_2=0.$

7. Перевозка из A_3 в B_4. Исключение A_3 и B_4

$a_3=50=b_4$ $\quad x_{34}=b_4,\ a_3=b_4=0.$

В общем случае количество элементов плана, вписываемых в макет методом «северо-западного угла», всегда равно $n+m-1$. При этом, разумеется, учитываются и подчеркнутые нули. Эти нули появляются в первоначальном и других планах, получаемых методом потенциалов, тогда, когда или есть пункты A_i с нулевыми запасами, или есть пункты B_j с нулевыми потребностями, или, наконец, в исходной ТЗ можно выделить замкнутую транспортную подзадачу. Наличие подчеркнутых нулей приводит, вообще говоря, к дополнительным вычислениям при поиске оптимального плана. Впрочем, от первых двух случаев, порождающих эти нули, легко избавиться простым выбрасыванием из макета соответствующих строк и столбцов. В полученной ТЗ уже все a_i и b_j будут положительными. Иногда так и поступают. Однако делать это не всегда целесообразно. Например, при многократном решении на ЭВМ конкретной ТЗ с фиксированной матрицей стоимостей и изменяющимися a_i и b_j $(i=\overline{1,m};\ j=\overline{1,n})$ проще всего каждый раз вводить в память лишь значения запасов и потребностей грузов, а матрицу стоимостей считывать с какого-либо носителя и не заботиться о ее преобразованиях.

Для дальнейшего изложения нам нужны некоторые понятия.

Клетки макета, соответствующие плану ТЗ, назовем базисными, а их совокупность — базисом.

Циклом L назовем последовательность клеток макета, обладающих свойствами:

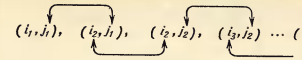


Рис. 39

а) первая и последняя клетки L совпадают;

б) каждые две соседние клетки расположены в одном ряду;

в) две первые клетки расположены в одном столбце;

г) никакие три клетки в одном ряду не располагаются.

На рисунке 39 показано схематическое изображение цикла. На рисунке 40 приведены некоторые разновидности циклов. Клетки в них заштрихованы и соединены отрезками прямых.

План, построенный методом «северо-западного угла», является ациклическим в том смысле, что из базисных клеток цикла не построишь. Однако если n, $m \geqslant 2$, то начиная с любой небазисной клетки можно построить, и притом единственный, цикл, все клетки которого, кроме начальной, будут базисными. Доказываются эти утверждения индукцией по $p = n + m$.

Остановимся еще на одном факте, на котором и базируется метод потенциалов. Формулируется он так.

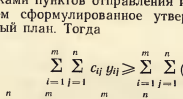
Если для некоторого плана $X = (x_{ij})$ ТЗ можно подобрать систему из $n+m$ чисел: $\alpha_1, \alpha_2, \ldots, \alpha_m$; $\beta_1, \beta_2, \ldots, \beta_n$, такую, что для небазисных клеток $\beta_j + \alpha_i \leqslant c_{ij}$, а для базисных $\beta_j + \alpha_i = c_{ij}$, то X — оптимальный план. Числа α_i и β_j называют соответственно потенциалами или платежами пунктов отправления и пунктов назначения.

Докажем сформулированное утверждение. Пусть $X^* = (y_{ij})$ — произвольный план. Тогда

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} y_{ij} \geqslant \sum_{i=1}^{m}\sum_{j=1}^{n} (\beta_j + \alpha_i) y_{ij} =$$

$$= \sum_{j=1}^{n} \beta_j \Big(\sum_{i=1}^{m} y_{ij}\Big) + \sum_{i=1}^{m} \alpha_i \Big(\sum_{j=1}^{n} y_{ij}\Big) = \sum_{j=1}^{n} \beta_j b_j + \sum_{i=1}^{m} \alpha_i a_i =$$

$$= \sum_{j=1}^{n} \beta_j \Big(\sum_{i=1}^{m} x_{ij}\Big) + \sum_{i=1}^{m} \alpha_i \Big(\sum_{j=1}^{n} x_{ij}\Big) = \sum_{i=1}^{m}\sum_{j=1}^{n} (\beta_j + \alpha_i) x_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij}.$$



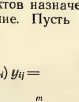


Рис. 40

Таким образом, стоимость перевозок любого плана (y_{ij}) не меньше стоимости плана X. Следовательно, X оптимален.

Теперь можно непосредственно перейти к описанию алгоритма решения ТЗ методом потенциалов.

Алгоритм метода потенциалов

1. Для замкнутой ТЗ (8) — (9) строим макет (табл. 6), где $x_{ij} = 0$ $(i = \overline{1, m};\ j = \overline{1, n})$.

2. Методом «северо-западного угла» формируем первоначальный план $X = (x_{ij})$ $(i = \overline{1, m};\ j = \overline{1, n})$, в котором ровно $n + m - 1$ элементов, каждый из которых или положительное число, или подчеркнутый нуль.

3. Вычисляем совокупность потенциалов α_i, β_j $(i = \overline{1, m},\ j = \overline{1, n})$ так, чтобы в любой базисной клетке выполнялось условие $\beta_j + \alpha_i = c_{ij}$. Для этого необходимо решить систему из $n + m - 1$ уравнений с $n + m$ неизвестными. Считаем $\alpha_1 = 0$ и последовательно определяем остальные потенциалы.

4. Проверяем текущий план на оптимальность. Если для всех клеток макета $\beta_j + \alpha_i \leqslant c_{ij}$ $(i = \overline{1, m};\ j = \overline{1, n})$, то план оптимален, и переходим к пункту 9.

5. Ищем в макете клетку (p, q), являющуюся началом будущего цикла перераспределения перевозок, для которой

$$\beta_q + \alpha_p - c_{pq} = \max_{i,\ j} (\beta_j + \alpha_i - c_{ij}).$$

6. Вычеркиваем из макета все ряды, кроме строки p и столбца q, имеющие не более одной базисной клетки. Этот процесс повторяем над оставшейся частью макета и т. д. до тех пор, пока еще есть что вычеркивать.

7. От клетки (p, q) строим цикл L. Все клетки L, кроме (p, q), должны быть базисными.

8. Пусть значением подчеркнутого нуля является нуль и $w = \min x_{ij}$, где $\min$ берется по всем клеткам, стоящим на четных местах в цикле. Организуем «сдвиг груза по циклу пересчета».

а) Во всех нечетных клетках перевозки увеличиваем на w:

$$x_{ij} = x_{ij} + w \qquad (10)$$

б) Во всех четных клетках перевозки уменьшаем на w:

$$x_{ij} = x_{ij} - w \qquad (11)$$

При этом все нули, получающиеся в результате пересчета (10) и (11), кроме первого, отмечаем как нули подчеркнутые. Тем самым во вновь полученном плане опять будет $n + m - 1$ базисных клеток.

Следующим выполняем шаг 3.

9. Выписываем из макета полученный оптимальный план в виде i, j, x_{ij} для всех клеток, где $x_{ij} > 0$. Подсчитываем стоимость этого плана. Вычисления прекращаем.

Отметим, как решать по предложенному алгоритму ТЗ, имеющие некоторые усложнения в постановке.

А1. **ТЗ с нарушенным балансом запасов и потребностей.**

Решение такой ТЗ простыми преобразованиями сводится к нахождению оптимального плана некоторой вспомогательной ТЗ. Об этом уже было сказано выше.

А2. **ТЗ с поиском максимума линейной формы *F*.**

Перейдем к форме $F_1 = -F$ и найдем план X^*, доставляющий ей минимум. Тогда X^* будет оптимальным и для исходной задачи, причем $\max F = -F_1(X^*)$.

А3. **ТЗ с запретными маршрутами.**

Речь идет о задачах, в которых нельзя перевозить груз из некоторых пунктов отправления A_i в некоторые пункты назначения B_j. В этом случае стоимости соответствующих перевозок полагаем равными достаточно большому числу. Тогда при отыскании оптимального плана соответствующие перевозки будут блокированы.

А4. **ТЗ с обязательными поставками.**

Иногда приходится решать ТЗ, в которых дополнительным условием в ограничениях является обязательное обеспечение конкретных перевозок по определенным маршрутам. В этом случае каждую обязательную перевозку $x_{ij} = d_{ij}$ реализуем условно, уменьшая на d_{ij} запасы в A_i и потребности в B_j. Если это сделать не удается, то исходная задача решения не имеет. В противном случае стоимости обязательных поставок полагаем равными достаточно большому числу, решаем полученную задачу и от ее оптимального плана переходим к оптимальному плану исходной ТЗ.

А5. **ТЗ с ограничениями снизу.**

Пусть требуется решить ТЗ, в которой некоторые из перевозок ограничены снизу $e_{ij} \leqslant x_{ij}$. Организуем условные перевозки, уменьшив на e_{ij} запасы в A_i и потребности в B_j. Если это сделать не удается, то исходная задача решения не имеет, в противном случае решаем полученную задачу и от ее оптимального плана переходим к оптимальному плану исходной ТЗ.

А6. **ТЗ с ограничениями сверху.**

Этот случай не является столь простым, как предыдущие, но и он сводится к определению решения некоторой вспомогательной замкнутой ТЗ (см. [20, 23]).

Программа «Транспорт». Рассмотренный ранее метод потенциалов решения ТЗ реализуется программой «Транспорт». Следует иметь в виду, что оптимизация может проводиться на максимум и минимум, замкнутость ТЗ не предполагается, но ограничения на переменные, кроме неотрицательности, не допускаются. Программа состоит из трех основных частей.

A. Ввод данных о конкретной ТЗ.

B. Преобразования по методу потенциалов.

C. Вывод оптимального плана.

Часть B (строки 450—1340) является ядром программы. Она оформлена в виде подпрограммы.

В соответствии с запросами на экране дисплея с пульта заносятся данные о конкретной ТЗ в такой последовательности:

1. Тип решаемой задачи (min — 1, max — 2).
2. Тип вывода (экран — 1, принтер — 2).
3. Количество пунктов отправления.
4. Количество пунктов назначения.
5. Данные о запасах груза в пунктах отправления A_i в виде:

запас в A_1, запас в A_2, ...

6. Данные о потребностях груза в пунктах назначения B_j в виде:

потребность в B_1, потребность в B_2, ...

7. Стоимости C_{ij} перевозок единицы груза из A_i в B_j. Эти данные представляют собой матрицу, элементы которой должны вводиться последовательно по строкам.

В зависимости от ответов пользователя на запросы решается ТЗ на min или max, и на экран или принтер выводятся результаты счета в следующем виде:

Оптимальный план:

откуда:	**куда:**	**сколько:**
i_1	j_1	x_1
i_2	j_2	x_2
...	...	...
i_s	j_s	x_s

MIN (MAX) $F = y$

время $= z$

Здесь:

i_k — номера пунктов отправления ($k = \overline{1, s}$);

j_k — номера пунктов назначения ($k = \overline{1, s}$);

x_k — перевозка по маршруту (i_k, j_k);

y — оптимальное значение F;

z — время в секундах решения задачи (без учета времени ввода-вывода).

Пример 13. Программа «Транспорт».

```
10 ' ТРАНСПОРТНАЯ ЗАДАЧА
20 '=============================ВВОД ДАННЫХ
30 SCREEN 0:COLOR 15,4,4:KEY OFF:B=1E+50
```

```
40 J1$="Т Р А Н С П О Р Т Н А Я  З А Д А Ч А
50 J2$="Запас A(I) в пункте отправления I:"
60 J3$="Заявка B(J) в пункте назначения J:"
70 J4$="Стоимость единицы перевозки из I в J
80 PRINT J1$:PRINT
90 INPUT "Задача (min  -1,max    -2)";O
100 INPUT "Вывод  (экран-1,принтер-2)";H
110 IF H=1 THEN OPEN "crt:" AS #1
120 IF H=2 THEN OPEN "lpt:" AS #1
130 INPUT"Количество пунктов отправления";S0
140 INPUT"Количество пунктов назначения ";P0
150 O=3-2*O:S=S0:P=P0:PRINT:L=S+1:M=P+1
160 DIM Z(L,M),X(L,M):'стоимости и перевозки
170 DIM A(L),B(M):PRINT J2$:PRINT
180 FOR I=1 TO S
190 PRINT "A(";I;")= ";:INPUT A(I)
200 NEXT I
210 PRINT:PRINT J3$:PRINT
220 FOR J=1 TO P
230 PRINT "B(";J;")= ";:INPUT B(J)
240 NEXT J
250 PRINT:PRINT J4$:PRINT
260 FOR I=1 TO S:FOR J=1 TO P
270 PRINT "Z(";I;",";J;")= ";
280 INPUT W:Z(I,J)=O*W
290 NEXT J:PRINT:NEXT I
300 GOSUB 450'=========================ЯДРО
310 H=TIME/50:CLS:U=0:'----------------ВЫВОД
320 PRINT #1,"ОПТИМАЛЬНЫЙ ПЛАН  :";CHR$(13)
330 PRINT #1,"откуда!куда!сколько"
```



```
340 PRINT #1,"-------------------"
350 FOR I=1 TO S0:FOR J=1 TO P0
360 IF X(I,J)<=0 THEN 390
370 U=U+X(I,J)*Z(I,J)
380 PRINT#1,"  ";I;"  ";J;"    ";X(I,J)
390 NEXT J:PRINT #1,CHR$(13)
400 NEXT I:PRINT #1,CHR$(13):U=ABS(U)
410 IF O=1 THEN PRINT #1,"MIN F =";U
420 IF O=-1 THEN PRINT #1,"MAX F =";U
430 PRINT #1,"время =";INT(H)+1
440 END:=========='имя программы "транс.пот"
450 '---------------------МЕТОД ПОТЕНЦИАЛОВ
460 TIME=0:CLS
470 LOCATE 8,9:PRINT "проводятся вычисления"
480 Q=S0+P0:DIM E(Q),F(Q)
490 '----------------------сведение к балансу
500 X=0:FOR I=1 TO S:X=X+A(I):NEXT I
510 Y=0:FOR J=1 TO P:Y=Y+B(J):NEXT J
520 IF X*Y=0 THEN 1340
530 IF X>Y THEN P=P+1:B(P)=X-Y
540 IF X<Y THEN S=S+1:A(S)=Y-X
550 '------------метод северо западного угла
560 I=1:J=1:H=1
570 R=A(I)-B(J)
580 IF (R<=0) AND (R<>0 OR B(J)=0) THEN 620
590 W=B(J):GOSUB 1260
600 IF J<P THEN J=J+1:GOTO 570
610 IF I<S THEN I=I+1:GOTO 570 ELSE 660
620 W=A(I):GOSUB 1290
630 IF I<S THEN I=I+1:GOTO 570
```

```
640 IF J<P THEN J=J+1:GOTO 570
650 '-------------------формирование платежей
660 T=B:GOSUB 1320:A(1)=0
 70 R=0:FOR H=1 TO Q:I=E(H):J=F(H)
680 IF X(I,J)=0 THEN 720
690 X=A(I):Y=B(J)
700 IF X=B AND Y<>B THEN A(I)=Z(I,J)-Y:R=1
710 IF X<>B AND Y=B THEN B(J)=Z(I,J)-X:R=1
720 NEXT H:IF R=1 THEN 670
730 '----------------проверка на оптимальность
740 R=0:FOR I=1 TO S:FOR J=1 TO P
750 F=A(I)+B(J)-Z(I,J)
760 IF F>R THEN R=F:X=I:Y=J
770 NEXT J,I:IF R=0 THEN 1340
780 '----------------------вычеркивание рядов
790 T=0:GOSUB 1320:FOR I=1 TO S:FOR J=1 TO P
800 IF X(I,J)<>0THEN A(I)=A(I)+1:B(J)=B(J)+1
810 NEXT J,I
820 R=0:FOR I=1 TO S
830 IF A(I)=-1 OR A(I)>1 OR I=X THEN 880
840 A(I)=-1:R=1:FOR J=1 TO P
850 IF B(J)=-1 OR X(I,J)=0 THEN 870
860 B(J)=B(J)-1:IF B(J)=0 THEN B(J)=-1
870 NEXT J
880 NEXT I
890 FOR J=1 TO P
900 IF B(J)=-1 OR B(J)>1 OR J=Y THEN 950
910 B(J)=-1:R=1:FOR I=1 TO S
920 IF X(I,J)=0 OR A(I)=-1 THEN 940
930 A(I)=A(I)-1:IF A(I)=0 THEN A(I)=-1
```



```
940 NEXT I
950 NEXT J:IF R=1 THEN 820
960 I=X:J=Y:W=B:'------------нахождение цикла
970 K=1:I=1
980 IF A(I)=-1 THEN I=I+1:GOTO 980
990 IF I=K OR X(I,J)=0 THEN I=I+1:GOTO 980
1000 R=X(I,J):IF W>R THEN W=R
1010 IF I<>X THEN K=J:J=1 ELSE 1050
1020 IF B(J)=-1 OR J=K THEN J=J+1:GOTO 1020
1030 IF X(I,J)=0THENJ=J+1:GOTO 1020 ELSE 970
1040 '----------------------перевозка по циклу
1050 I=X:J=Y:V=0:IF W=-1 THEN W=0
1060 K=I:I=1
1070 IF A(I)=-1 THEN I=I+1:GOTO 1070
1080 IF I=K OR X(I,J)=0 THEN I=I+1:GOTO 1070
1090 R=X(I,J):IF R>=0 THEN 1110
1100 GOSUB 1220:X(I,J)=0:X(X,Y)=-1:GOTO 660
1110 IF R>W THEN X(I,J)=R-W:GOTO 1140
1120 IF V<>0 THEN X(I,J)=-1
1130 IF V=0 THEN GOSUB 1220:X(I,J)=0:V=1
1140 IF I=X THEN X(X,Y)=W:GOTO 660
1150 IF I<>X THEN K=J:J=1
1160 IF B(J)=-1 THEN J=J+1:GOTO 1160
1170 IF J=K OR X(I,J)=0 THEN J=J+1:GOTO 1160
1180 IF X(I,J)<=0 THEN 1200
1190 X(I,J)=X(I,J)+W:GOTO 1060
1200 IF W>0 THENX(I,J)=W:GOTO 1060 ELSE 1090
1210 '--------------------------подпрограммы
1220 FOR H=1 TO Q
1230 IF E(H)<>I OR F(H)<>J THEN 1250
```

```
1240 E(H)=X:F(H)=Y:H=Q
1250 NEXT H:RETURN:'----------------------
1260 GOSUB 1310
1270 E(H)=I:F(H)=J:H=H+1:A(I)=R:B(J)=0
1280 RETURN
1290 GOSUB 1310:E(H)=I
1300 F(H)=J:H=H+1:A(I)=0:B(J)=-R:RETURN'----
1310 IF W<>0 THEN X(I,J)=W:GOTO 1313
1312 IF W=0 THEN X(I,J)=-1
1313 RETURN
1320 FOR I=1 TO S:A(I)=T:NEXT
1330 FOR J=1 TO P:B(J)=T:NEXT
1340 RETURN:'=====имя программы  "транс.пот"
```

Приведем несколько примеров работы с программой «Транспорт».

Пример 14. Найти план, минимизирующий расходы от перевозок для следующей задачи.

Запасы груза в складах A_i: 20, 30, 40, 10.

Потребности в грузе в магазинах B_j: 17, 15, 12, 8, 3, 4.

Матрица транспортных расходов на перевозку единицы продукции от склада A_i в магазин B_j:

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	2	3	4	2	1	3
A_2	2	4	1	1	2	1
A_3	3	1	1	4	2	1
A_4	1	1	2	2	2	3

Вводимые данные:

1, 2, 4, 6
20, 30, 40, 10
17, 15, 12, 8, 3, 4
2, 3, 4, ..., 3, 1, 1, ..., 2, 3

Сообщение: Смотри слева на схеме 1.

Пример 15. Найти план в предыдущей задаче, максимизирующий расходы от перевозок.

Вводимые данные:

2, 2, 4, 6.

Далее — как и в примере 14.

Сообщение: Смотри справа на схеме 1.

Схема 1

Оптимальный план: откуда! куда! сколько!			Оптимальный план: откуда! куда! сколько!		
1	1	7	1	3	12
1	5	3	1	6	4
2	3	12	2	2	15
2	4	8	2	5	3
2	6	4	3	1	17
3	2	15	3	4	8
4	1	10			
MIN F=66 Время=34			MAX F=209 Время=42		

Обратим внимание, насколько разнятся планы в двух последних задачах по расходам на перевозки: 66 и 209. Этот факт является косвенным подтверждением того, что «угадать» оптимальный план, не используя специальные алгоритмы (программы), даже для небольших задач весьма сложно.

В заключение этого пункта отметим наличие отличных от метода потенциалов весьма эффективных алгоритмов решения ТЗ. К ним относится так называемый «венгерский алгоритм», основные идеи которого обсуждаются в следующем пункте на примере частной задачи транспортного типа.

2.3. Задача о назначениях

Постановка задачи. Алгоритм. Одна из распространенных формулировок задачи о назначениях (ЗН) выглядит так.

Пусть имеется m работников A_1, A_2, ..., A_m и n должностей B_1, B_2, ..., B_n. Известна мера c_{ij} полезности (эффективности, ценности, стоимости) работника A_i на должности B_j ($i=\overline{1, m}$; $j=\overline{1, n}$). Требуется организовать такое закрепление работников на должности, при котором суммарная стоимость назначений будет максимальной.

Если c_{ij} интерпретировать как издержки назначения A_i на B_j, то естественно решать задачу закрепления работников на должности, при котором минимизируются общие затраты.

Построим математическую модель этой задачи.

Пусть $X_{ij} = \begin{cases} 1, & \text{если } A_i \text{ назначается на } B_j \\ 0, & \text{если } A_i \text{ не назначается на } B_j. \end{cases}$

Рассмотрим булеву матрицу (x_{ij}) размера $m \times n$, такую, что:

$$\sum_{i=1}^{m} x_{ij} \leqslant 1 \ (j=\overline{1, n}), \ \sum_{j=1}^{n} x_{ij} \leqslant 1 \ (i=\overline{1, m}) \quad \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} \leqslant \min(m, n) \qquad (1)$$

При этих условиях (x_{ij}) называется планом или матрицей назначений. Среди планов выделяют такие, для которых в (1) достигается равенство. Они называются насыщенными.

Стоимость любого плана (x_{ij}) выражается суммой $F=\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$.

Окончательно математическая модель задачи будет такой.

Найти матрицу $X=(x_{ij})$ $(i=\overline{1,\ m};\ j=\overline{1,\ n})$, такую, что:

$$F(X)=\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \to \text{opt (max или min)} \qquad (2)$$

при условиях

$$\sum_{i=1}^{m} x_{ij} \leqslant 1 \ (j=\overline{1,\ n});\ \sum_{j=1}^{n} x_{ij} \leqslant 1 \ (i=\overline{1,\ m}) \qquad (3)$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}=\min\{n,\ m\} \qquad (4)$$

$$x_{ij} \in \{0,\ 1\} \ (i=\overline{1,\ m};\ j=\overline{1,\ n}) \qquad (5)$$

Другими словами, ищется насыщенная матрица назначений, оптимизирующая форму F.

Фактически мы получили задачу линейного программирования, но с булевыми переменными. Однако если отбросить ограничение (5), заменив его более простым условием неотрицательности x_{ij}, то задача о назначениях превращается в частный случай транспортной задачи, где все запасы a_i $(i=\overline{1,\ m})$ и потребности b_j $(j=\overline{1,\ n})$ равны единице. Если решать эту задачу любым методом, приводящим при целых a_i и b_j к целочисленному оптимальному решению, то неучтенные условия (5) удовлетворяются автоматически. Таким образом, для решения задачи о назначениях оказывается пригодным рассмотренный ранее метод потенциалов. Необходимо лишь положить там $a_i=1$, $b_j=1$ $(i=\overline{1,\ m};\ j=\overline{1,\ n})$.

Но именно эти особенности задачи о назначениях позволяют разработать для нее более эффективные модификации соответствующих алгоритмов. Ниже приводится подобная модификация «венгерского алгоритма».

Предварительно сделаем несколько замечаний. Решение исходной ЗН всегда можно свести в некотором смысле к эквивалентной ЗН с одинаковым количеством работников и должностей.

Пусть $p=\max(n,\ m)$ $(n \neq m)$.

Рассмотрим возможные случаи.

1) $m<n$. Добавим к работникам $n-m$ «подставных» лиц, для которых цены назначения на любые должности равны нулю. Найдя оптимальный план этой ЗН и не учитывая в нем подставных лиц, получим оптимальное назначение исходной ЗН.

2) $m>n$. Добавим к имеющимся должностям $m-n$ условных мест, цены назначения на которые положим равными нулю. Найдя оптимальный план этой ЗН и не учитывая в нем работников, назначенных на несуществующие должности, получим оптимальный план исходной задачи.

Таким образом, всегда можно считать $n=m=p$.

Описание алгоритма будем проводить с использованием специального макета (см. табл. 9). Поскольку план $X=(x_{ij})$ мы будем по-

Таблица 9

Макет задачи о назначениях

	B_1	B_2	...	B_p	E_2	E_4
A_1	c_{11}	c_{12}	...	c_{1p}	i_1	s_1
A_2	c_{22}	c_{22}	...	c_{2p}	i_2	s_2
...	...	...	...	...	...	...
A_p	c_{p1}	c_{p2}	...	c_{pp}	i_p	s_p

E_1	j_1	j_2	...	j_p
E_3	t_1	t_2	...	t_p

следовательно формировать в основной матрице макета, где первоначально проставлены стоимости, то для однозначной идентификации значений нам придется ввести для клеток три типа нулей.

0 — простой нуль (текущая стоимость — 0, значение компоненты плана — 0);

0* — нуль со звездочкой (текущая стоимость — 0, значение компоненты плана — 1);

0′ — нуль со штрихом (текущая стоимость — 0, значение компоненты плана — 0, но при следующем улучшении плана оно может обратиться в 1). В программе «Назначение» эти нули кодируются соответственно символами: 0, —1, —2.

Одномерные массивы E_1 — E_4 являются вспомогательными и служат для указания в основной матрице адресов определенных элементов. Они будут заполняться в процессе вычислений. Их значения интерпретируются так.

$E_1(j)$ $(j=\overline{1,\ p})$ — номер строки в столбце j, где расположен 0. Если $E_1(j)=0$, то подобные нули отсутствуют.

$E_2(i)$ $(i=\overline{1,\ p})$ — номер столбца в строке i, где расположен 0*. Если $E_2(i)=0$, то подобные нули отсутствуют.

$E_4(i)$ $(i=\overline{1,\ p})$ — номер столбца в строке i, где расположен 0′. Если $E_4(i)=0$, то подобные нули отсутствуют.

$E_3(j)$ $(j=\overline{1,\ p})$ — метка столбца j. Столбец считается занятым при $E_3(j)\neq 0$ и свободным при $E_3(j)=0$.

Теперь приступим к описанию обещанной модификации «венгерского алгоритма» задачи о назначениях.

Алгоритм решения задачи о назначениях (модификация «венгерского алгоритма»).

1. **Формирование макета.** Для ЗН (2) — (5) строим макет (табл. 9), где $p=\max(m,\ n)$, если решается задача на максимум, меняем знаки у C_{ij}. Элементы массивов E_1 — E_3 полагаем равными нулю.

2. **Предварительное преобразование матрицы стоимостей C.**

а) Из каждого элемента столбца j $(j=\overline{1,\ p})$ вычитаем его наименьший элемент: $C_{ij}=C_{ij}-\min\limits_{1\leqslant x\leqslant p} C_{xj}$ $(i=\overline{1,\ p})$.

б) Из каждого элемента строки i $(i=\overline{1,\ p})$ вычитаем ее наименьший элемент: $C_{ij}=C_{ij}-\min\limits_{1\leqslant s\leqslant p} C_{is}$ $(j=\overline{1,\ p})$.

Полученная матрица C имеет по крайней мере один нуль в каждой строке и каждом столбце. Суммируем все находимые в процессе счета минимумы c_{ij} по строкам и столбцам и полученное значение присваиваем величине F.

3. **Формирование начального назначения.** Просматриваем поочередно слева направо каждую i-ю строку $(i=\overline{1,\ p})$ матрицы стоимостей $C=(c_{ij})$. Если встретится нуль и в его столбце отсутствует элемент 0*, то метим 0 звездочкой. В каждом ряду (строке и столбце) должно быть не более одного элемента 0*. Одновременно создаем массивы адресных ссылок E_1 и E_2 и массив занятости столбцов E_3. Делаем это так. Если $(i,\ j)$ — клетка расположения в макете 0*, то $E_1(j)=i$, $E_2(i)=j$, $E_3(j)=1$. Подсчитываем в C количество H элементов 0* и переходим к пункту 9.

4. Все строки объявляем свободными: $E_4(i)=0$ $(i=\overline{1,\ p})$. В дальнейшем будут появляться и занятые строки $(E_4(i)\neq 0)$. Элементы матрицы $C=(c_{ij})$, стоящие на пересечении свободных рядов, называются свободными. Остальные элементы C — занятые.

5. Если в C нет свободных нулей, то переходим к пункту 8.

6. **Разметка макета.** Просматриваем поочередно слева направо строки матрицы C, выбираем в ней первый свободный нуль и метим его штрихом. Пусть $(i,\ j)$ — местоположение в макете нового 0′. Фиксируем его адрес: $E_4(i)=j$. Тем самым строка i объявляется занятой. Если в ней нет элементов 0*, то переходим к пункту 7.

Пусть $(i,\ s)$ — местоположение в строке i элемента 0*. Снимаем метку занятости столбца S: $E_3(S)=0$ и переходим к пункту 5.

7. **Дополнительные назначения.** Начиная с только что отмеченного нуля, строим цепочку: от данного 0′ по столбцу к 0*, от него по строке к 0′ и т. д. Поиск соответствующих элементов облегчают адресные массивы E_1, E_2 и E_4. Цепочка оборвется на 0′. Возможно, что она вообще будет состоять из одного начального 0′. Заменяем в цепочке каждый 0* на 0 и 0′ на 0*. В процессе этой замены актуализируем значения элементов массивов E_1, E_2 и E_3. Количество H элементов 0* увеличиваем на 1: $H=H+1$. Переходим к пункту 9.

8. **Изменение стоимостей.** Пусть w — минимальный среди всех свободных элементов матрицы C. Преобразуем C так. Элементы свободных строк уменьшим на w, а элементы занятых столбцов увеличим на w. При этом 0* и 0′ оставляем без изменения. Эти операции приводят к появлению в C свободных нулей.

Пусть изменениям подверглись L_1 строк и L_2 столбцов. Тогда текущее значение F стоимости плана вычисляем следующим образом: $F=F+(L_1-L_2)*w$ и переходим к пункту 6.

9. **Проверка окончания счета и вывод результатов.** Если количество H отмеченных звездочкой нулей меньше P, то переходим к пункту 4, иначе процесс окончен. Места, занимаемые элементами 0*, соответствуют переменным x_{ij}, равным единице. Формируем и выводим оптимальный план $X=(x_{ij})$ исходной задачи:

$$X_{ij}=\begin{cases}1, & \text{если } c_{ij} \text{ есть } 0^*\\ 0, & \text{если } c_{ij} \text{ не есть } 0^*\end{cases} \qquad (i=\overline{1,\ m};\ j=\overline{1,\ n}).$$

Выводим значение стоимости плана X:

$$F(x)=\begin{cases}F \text{ для задачи на минимум}\\ -F \text{ для задачи на максимум.}\end{cases}$$

Вычисления прекращаем.

Непосредственное обоснование алгоритма предлагаем провести читателю самостоятельно.

Заметим следующее. Если в ЗН имеются несовместимые пары $(A_i,\ B_j)$, то есть назначение A_i на должность B_j невозможно, то для ее решения ничего нового придумывать не требуется. Достаточно соответствующую стоимость задать так:

$$C_{ij}=\begin{cases}B \text{ для задачи на минимум}\\ -B \text{ для задачи на максимум,}\end{cases}$$

где B — достаточно большое число.

Программа «Назначение». Приведенный в предыдущем разделе алгоритм решения ЗН (2) (5) реализуется программой «Назначение». Оптимизация по ней может проводиться на максимум и минимум. Равенство количества работников и должностей не предполагается. Структура программы соответствует структуре алгоритма.

По запросам с экрана дисплея с пульта заносятся данные о конкретной ЗН в такой последовательности:

1. Тип задачи (min —1, max —2).
2. Тип вывода (экран —1, принтер —2).
3. Количество работников.
4. Количество должностей.
5. Цена назначения (работник — должность). Эти данные представляют собой матрицу, элементы которой должны вводиться последовательно по строкам.

В зависимости от ответов пользователя на запросы решается ЗН на min или max и на экран или принтер выводятся результаты счета в следующем виде.

Назначение

работник: должность

i_1 j_1
i_2 j_2
... ...
i_q j_q

MIN (MAX) $F=y$
время$=z$

Здесь:

$q=\min(m; n)$;

i_k — номера работников ($k=\overline{1, q}$);

j_k — номера должностей ($k=\overline{1, q}$);

y — оптимальное значение формы F;

z — время в секундах решения ЗН (без учета времени ввода-вывода)

Пример 16. Программа «Назначение».

```
10 '              ЗАДАЧА О НАЗНАЧЕНИЯХ
20 '     (МОДИФИКАЦИЯ ВЕНГЕРСКОГО АЛГОРИТМА)
30 '============================ВВОД ДАННЫХ
40 SCREEN 0:COLOR 15,4:WIDTH 40:KEY OFF
50 PRINT "ЗАДАЧА  О  НАЗНАЧЕНИЯХ":PRINT
60 INPUT "Тип задачи (min  -1,max     -2) ";O
70 INPUT "Тип вывода (экран-1,принтер-2) ";H
80 IF H=1 THEN OPEN "crt:" AS #1
90 IF H=2 THEN OPEN "lpt:" AS #1
100 INPUT "Количество работников          ";S1
110 INPUT "Количество должностей          ";S2
120 O=3-2*O:B=1E+50
130 IF S1<S2 THEN P=S2 ELSE P=S1
140 DIM Z(P,P),A(P),B(P),C(P),D(P):PRINT
150 PRINT "Издержки назначения":PRINT
160 FOR I=1 TO S1
170 FOR J=1 TO S2:PRINT "Z(";I;",";J;")= ";
180 INPUT W:Z(I,J)=O*W
190 NEXT J:PRINT:NEXT I
200 CLS:TIME=0:'=======================ЯДРО
210 LOCATE 8,9:PRINT "проводятся вычисления"
220 '-------------------начальное назначение
230 IF S1<S2 THEN 300
240 FOR J=1 TO S2:W=Z(1,J)
250 FOR I=2 TO P:IF Z(I,J)<W THEN W=Z(I,J)
260 NEXT I
270 FOR I=1 TO P
280 Z(I,J)=Z(I,J)-W:NEXT I:Z=Z+W
290 NEXT J
300 IF S2<S1 THEN 360
310 FOR I=1 TO S1:W=Z(I,1):FOR J=2 TO P
320 IF Z(I,J)<W THEN W=Z(I,J)
330 NEXT J
340 FOR J=1 TO P:Z(I,J)=Z(I,J)-W:NEXT J
350 Z=Z+W:NEXT I
```

```
360 FOR I=1 TO P:FOR J=1 TO P
370 IF A(J)<>0 THEN 400
380 IF Z(I,J)<>0 THEN 400
390 Z(I,J)=-1:A(J)=I:B(I)=J:J=P:U=U+1
400 NEXT J,I:IF U=P THEN 670
410 '-------------------------------разметка
420 FOR J=1 TO P:C(J)=A(J):D(J)=0:NEXT
430 R=0:FOR J=1 TO P:IF C(J)<>0 THEN 470
440 FOR I=1 TO P
450 IF D(I)=0 AND Z(I,J)=0 THEN X=I:Y=J:I=P:
460 NEXT I                                  J=P:R=1
470 NEXT J:IF R=0 THEN 570
480 Z(X,Y)=-2:IF B(X)=0 THEN B(X)=Y:GOTO 510
490 D(X)=Y:C(B(X))=0:GOTO 430
500 '---------------дополнительные назначения
510 Z(X,Y)=-1:R=A(Y):A(Y)=X:IF R=0 THEN 530
520 X=R:SWAP B(X),D(X):Y=B(X):GOTO 510
530 FOR I=1 TO P
540 J=D(I):IF J<>0 THEN Z(I,J)=0
550 NEXT I:U=U+1:IF U<P THEN 420 ELSE 680
560 '--------------------изменение стоимостей
570 H=B:FOR I=1 TO P:FOR J=1 TO P
580 IF C(J)<>0 OR D(I)<>0 THEN 600
590 IF H>Z(I,J) THEN H=Z(I,J):X=I:Y=J
600 NEXT J,I:W=0
610 FOR I=1 TO P:IF D(I)<>0 THEN 630
620 FOR J=1 TO P:Z(I,J)=Z(I,J)-H:NEXT:W=W+1
630 NEXT I
640 FOR J=1 TO P:IF C(J)=0 THEN 660
```



```
650 FOR I=1 TO P:Z(I,J)=Z(I,J)+H:NEXT:W=W-1
660 NEXT J:Z=Z+HXW:GOTO 430
670 '=================================ВЫВОД
680 T=TIME/50:CLS
690 PRINT #1,"Н А З Н А Ч Е Н И Е";CHR$(13)
700 PRINT #1,"работник!должность"
710 PRINT #1,"---------!---------"
720 FOR I=1 TO S1:FOR J=1 TO S2
730 IF Z(I,J)<>-1 THEN 750
740 PRINT#1,"  ";I;"    !    ";J
750 NEXT J,I:PRINT #1,CHR$(13):Z=Z*0
760 IF O=1 THEN PRINT #1,"MIN F =";Z
770 IF O=-1 THEN PRINT #1,"MAX F =";Z
780 PRINT#1,"время =";INT(T)+1
790 END:'--------'имя программы "назнач.вен"
```

Приведем два примера работы с программой «Назначение».

Пример 17. Пусть имеется 5 типов работ и 5 механизмов для их выполнения, причем каждый механизм может использоваться на любой работе. Производительность механизма A_i на работе B_j $(i, j=\overline{1, 5})$ известна и задается следующей матрицей:

	B_1	B_2	B_3	B_4	B_5
A_1	3	2	5	4	5
A_2	5	3	2	1	4
A_3	1	2	6	3	2
A_4	4	3	5	4	1
A_5	1	2	2	1	5

Требуется распределить механизмы по работам так, чтобы суммарная производительность была максимальной.

Ясно, что это типичная задача о назначениях.

Вводимые данные:

2, 1, 5, 5
3, 2, ..., 5 (матрица цен).

Сообщение (смотри слева на схеме 2):

Пример 18. Найти оптимальный план, минимизирующий издержки назначения для задачи, исходные данные которой задаются следующей матрицей цен:

	B_1	B_2	B_3	B_4	B_5	B_6	B_7
A_1	10	2	3	2	4	4	1
A_2	2	10	3	8	2	1	7
A_3	1	5	10	2	1	3	3
A_4	1	4	1	10	7	2	5

Вводимые значения:

1, 1, 4, 7
10, 2, ..., 5 (матрица цен)

Сообщение (смотри справа на схеме 2):

Схема 2

Назначение	
работник	должность
1	4
2	1
3	3
4	2
5	5

MAX F = 23
время = 2

Назначение	
работник	должность
1	7
2	6
3	1
4	3

MIN F = 4
время = 2

Остановимся на некоторых характеристиках программы «Назначение». Во-первых, в ней используются всего 4 линейных рабочих массива, что позволяет размещать в оперативной памяти довольно «большие» задачи. Во-вторых, быстродействие программы также вполне приемлемо. На ЭВМ «Ямаха» ЗН размером 45×45 выполняются за 20—40 минут. Упомянем об одной разновидности постановки задачи о назначениях.

Пусть имеется m групп лиц A_i по a_i ($i=\overline{1, m}$) человек в каждой и n типов работ B_j по b_j ($j=\overline{1, n}$) единиц в каждой. Известна производительность c_{ij} лица i-й группы при выполнении j-го типа работ. Необходимо определить, сколько лиц, из какой группы и на какую категорию работ назначить, чтобы суммарная производительность была максимальной.

Обозначим через x_{ij} количество лиц i-й группы, назначаемых для выполнения работ j-го типа. Тогда математическая модель задачи будет такой.

Найти план назначения $X=(x_{ij})$, максимизирующий значение линейной функции $\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$ при ограничениях:

1. $\sum_{j=1}^{n} x_{ij}=a_i$ ($i=\overline{1, m}$). 2. $\sum_{i=1}^{m} x_{ij}=b_{ij}$ ($j=\overline{1, n}$).

3. x_{ij} — целые неотрицательные числа.

Фактически мы получили модель транспортной задачи, но с целочисленными переменными. Поскольку алгоритм потенциалов и «венгерский метод» решения ТЗ при целых a_i ($i=\overline{1, m}$) и b_j ($j=\overline{1, n}$) дают целочисленное решение, то сформулированную ЗН можно решать любым из этих методов.

3. КОРНИ МНОГОЧЛЕНОВ

Пусть n — натуральное число и

$$f(z)=a_0z^n+a_1z^{n-1}+\ldots+a_n \quad (a_0\neq 0) \tag{1}$$

полином, или, по-другому, многочлен, комплексного переменного с комплексными коэффициентами a_k ($k=\overline{0, n}$). В соответствии с основной теоремой алгебры $f(z)$ имеет ровно n корней (нулей):

$$w_1, w_2, \ldots, w_n. \tag{2}$$

До появления компьютеров нахождение всех величин w_k ($k=\overline{1, n}$) считалось весьма трудоемкой задачей и для каждого конкретного случая опиралось, как правило, на выбор специальных приемов и методов, учитывающих свойства $f(z)$. В настоящее время разработаны весьма простые и эффективные алгоритмы нахождения корней полиномов, ориентированные на использование ЭВМ. Три таких алгоритма и будут рассмотрены ниже. Все они являются итерационными и весьма просты в реализации.

3.1. Методы Ньютона и секущих

Рассматриваемые методы обеспечивают довольно быструю сходимость при наличии у многочлена корней не выше второй кратности.

В методе Ньютона нуль w_s функции $f(z)$ находится, исходя из начального приближения z_0, как предел последовательности комплексных чисел z_k:

$$z_{k+1}=z_k-f(z_k)/f'(z_k) \quad (k=0, 1, 2, \ldots) \tag{3}$$

при условии $f'(z_k)\neq 0$.

В методе секущих нуль w_s функции $f(z)$ находится, исходя из начальных приближений z_0 и z_1, как предел последовательности комплексных чисел z_k:

$$z_{k+1}=z_k-f(z_k)/[(f(z_k)-f(z_{k-1}))/(z_k-z_{k-1})] \quad (k=1, 2, 3, \ldots) \tag{4}$$

В каком случае итерации по формулам (3) или (4) следует приостанавливать? Как правило, это делают тогда, когда для очередного z_k выполняется условие $|f(z_k)| < \varepsilon$, где $\varepsilon > 0$ — заранее фиксированное малое положительное число. При этом считают, что $w_s \approx z_k$. Далее, после понижения степени f: $f(z) = f(z)/(z - z_k)$ определяют другие корни исходного полинома. Учитывая крайнюю неустойчивость корней w_s ($s = \overline{1, n}$) некоторых полиномов от их коэффициентов, целесообразно отыскивать величины w_s различными способами. Это позволит организовать анализ полученных приближений к w_s ($s = \overline{1, n}$) и уточнение последних.

В программе примера 1 реализована комплексная арифметика. По ней, в соответствии с диалогом, корни $f(z)$ могут находиться и методом Ньютона, и методом секущих.

Пример 1. Программа нахождения корней полиномов с вещественными или комплексными коэффициентами методами Ньютона и секущих.

```
10 '        НАХОЖДЕНИЕ КОРНЕЙ ПОЛИНОМОВ
20 '          МЕТОДЫ НЬЮТОНА И СЕКУЩИХ
30 '-------------------------------------------------
40 '    F(Z)= a(0)*z^n + a(1)*z^(n-1)+ ...+
50 '                     a(n-1)*z+ a(n)
60 '1) a(k)=A(k)        - коэфф. вещественные
70 '                     (k=0,1...n)
80 '2) a(k)=A(k)+i*B(k)- коэфф. комплексные
90 '                     (k=0,1...n)
100 '------------------------------------------------
110 SCREEN 0:COLOR 15,4:KEY OFF:E2=1E-05
120 A$="##)######.#######  +######.########*i"
130 D$="3. вещественные части коэффициентов:
140 E$="4. комплексные части коэффициентов:"
150 F$="3. коэффициенты:"
160 '=================================ВВОД
170 PRINT "   КОРНИ МНОГОЧЛЕНОВ":PRINT
180 INPUT "1. степень многочлена ";N0
```



```
190 R=N0+1:DIM C(R),D(R),A(R),B(R)
200 PRINT "2. тип коэффициентов:"
210 PRINT "   вещественные - 1"
220 INPUT "   комплексные  - 2   ";T:PRINT
230 IF T=1 THEN PRINT F$ ELSE PRINT D$
240 FOR K=0 TO N0
250 PRINT "   A(";K;") = ";:INPUT C(K)
260 NEXT
270 IF T=2 THEN PRINT E$ ELSE 310
280 FOR K=0 TO N0
290 PRINT "   B(";K;") = ";:INPUT D(K)
300 NEXT
310 PRINT
320 IF T=1 THEN PRINT "4."; ELSE PRINT "5.";
330 INPUT" 'точность' вычислений ";E1:E=E1^2
340 PRINT:PRINT "В О З М О Ж Н Ы Е  МЕТОДЫ:"
350 PRINT:PRINT "   НЬЮТОНА       --- 1"
360 PRINT "   СЕКУЩИХ       --- 2"
370 INPUT "   выход         --- 3  ";O:CLS
380 LOCATE 8,0:PRINT "проводятся вычисления"
390 PRINT:LOCATE 3,4:ON O GOTO 400,410,920
400 PRINT "метод НЬЮТОНА":GOTO 420
410 PRINT "метод СЕКУЩИХ"
420 N=N0:W=1
430 FOR S=0 TO N:A(S)=C(S):B(S)=D(S):NEXT
440 LOCATE 3,6:PRINT "корни многочлена:"
450 PRINT:T=TIME:IF O=2 THEN 560'========ЯДРО
460 GOSUB 860:'----------------метод НЬЮТОНА
470 GOSUB 760:R=C^2+D^2
480 IF R<E THEN GOSUB 870:GOTO 920
490 GOSUB 720:L=(A*C+B*D)/R:M=(B*C-A*D)/R
```

```
500 IF ABS(L)+ABS(M)<=E1 THEN 520
510 X=X-L:Y=Y-M:GOTO 470
520 PRINTUSING A$;W,X,Y
530 N=N-1:W=W+1:GOSUB 820
540 IF N>0 THEN 460 ELSE GOSUB 890:GOTO 340
550 '----------------------------метод СЕКУЩИХ
560 U=0:V=0:C=A(N):D=B(N)
570 GOSUB 860
580 GOSUB 720
610 R=L1^2+M1^2
620 IF R<E THEN GOSUB 870:GOTO 920
630 L=(A*L1+B*M1)/R
640 M=(B*L1-A*M1)/R:R=ABS(L)+ABS(M)
650 IF R<=E1 THEN 670
660 U=X:V=Y:C=A:D=B:X=X-L:Y=Y-M:GOTO 580
670 PRINTUSING A$;W,X,Y
680 N=N-1:W=W+1:GOSUB 820
690 IF N>0 THEN 560 ELSE GOSUB 890:GOTO 340
700 '-----------------------------подпрограммы
710 '----------вещественная и мнимая части F
720 A=A(0):B=B(0)
730 FOR K=1 TO N
740 L=A*X-B*Y+A(K):B=B*X+A*Y+B(K):A=L
750 NEXT:RETURN
760 C=A(0)*N:D=B(0)*N:IF N=1 THEN 810
770 FOR K=1 TO N-1
780 L=C*X-D*Y+(N-K)*A(K)
790 D=D*X+C*Y+(N-K)*B(K):C=L
800 NEXT
810 RETURN:'----------------понижение степени F
```



```
820 FOR K=1 TO N:L=K-1
830 A(K)=A(K)+A(L)*X-B(L)*Y
840 B(K)=B(K)+B(L)*X+A(L)*Y
850 NEXT:RETURN:'-----------------------------
860 X=RND(-TIME)*3:Y=RND(-TIME)*3+1:RETURN
870 PRINT "приближение к кратному корню"
880 PRINT:PRINTUSING A$;W,X,Y:RETURN:'-------
890 PRINT
900 PRINT "время счета = ";(TIME-T)/50;"сек"
910 RETURN
920 END:'==========имя программы "КОРНИ-1"
```

Пример 2. По программе примера 1 методом Ньютона найти приближенные значения корней полинома $f(z)=z^4-4z^3+5z^2-4z+4$.

Соответствующие диалог и сообщение о результатах счета ниже.

ДИАЛОГ

```
К О Р Н И   М Н О Г О Ч Л Е Н О В

1. степень многочлена ? 4
2. тип коэффициентов:
   вещественные - 1        комплексные - 2   ? 1
3. коэффициенты:
   A( 0 ) = ? 1
   A( 1 ) = ? -4    A( 3 ) = ? -4
   A( 2 ) = ? 5     A( 4 ) = ? 4
4. 'точность' вычислений ? 0.0000001

В О З М О Ж Н Ы Е   М Е Т О Д Ы:

  НЬЮТОНА    --- 1     СЕКУЩИХ    --- 2
  выход      --- 3  ? 1
     СООБЩЕНИЯ И РЕЗУЛЬТАТЫ ВЫЧИСЛЕНИЙ
        проводятся вычисления
  метод НЬЮТОНА
```

```
корни многочлена:
1)     0.0000000        +1.0000000*i
2)     2.0000000        +0.0000003*i
3)     2.0000000        -0.0000003*i
4)     0.0000000        -1.0000000*i
время счета =  28.9 сек
В О З М О Ж Н Ы Е  МЕТОДЫ:
НЬЮТОНА      --- 1          СЕКУЩИХ      --- 2
выход        --- 3  ?
```

Отметим следующее обстоятельство. Поскольку начальная точка для последовательных приближений генерируется случайным образом, то повторный прогон того же самого примера может привести как к иному времени счета, так и к небольшим отклонениям от полученных значений корней.

Пример 3. По программе примера 1 методом секущих найти приближенное значение корней полинома $f(z)=z^3-z^2-iz+i$.

Соответствующие диалог и сообщение о результатах счета ниже.

```
ДИАЛОГ
      К О Р Н И  М Н О Г О Ч Л Е Н О В
1. степень многочлена ? 3
2. тип коэффициентов:
   вещественные - 1    комплексные - 2    ? 2
3. вещественные части коэффициентов:
   A( 0 ) = ? 1          A( 2 ) = ? 0
   A( 1 ) = ? -1         A( 3 ) = ? 0
4. комплексные части коэффициентов:
   B( 0 ) = ? 0          B( 2 ) = ? -1
   B( 1 ) = ? 0          B( 3 ) = ? 1
5. 'точность' вычислений ? 0.0000001
В О З М О Ж Н Ы Е  МЕТОДЫ:
   НЬЮТОНА      --- 1     СЕКУЩИХ      --- 2
   выход        --- 3  ? 2
СООБЩЕНИЯ И РЕЗУЛЬТАТЫ ВЫЧИСЛЕНИЙ
```



```
.проводятся вычисления
   метод СЕКУЩИХ
   корни многочлена:
1)     1.0000000        +0.0000000*i
2)     0.7071068        +0.7071068*i
3)    -0.7071068        -0.7071068*i
время счета =  13.02 сек
В О З М О Ж Н Ы Е  МЕТОДЫ:
   НЬЮТОНА      --- 1     СЕКУЩИХ      --- 2
   выход        --- 3  ?
```

3.2. Метод «обруча»

Приведенный ниже метод нахождения корней w_s ($s=\overline{1,\ n}$) многочлена $f(z)$ с успехом можно применять вне зависимости от порядка кратности w_s. Он опирается на тот факт, что $f(z)$ — аналитическая во всей комплексной плоскости функция, и на принцип максимума модуля аналитической функции. Этот принцип формулируется так.

Модуль функции $f(z)$, аналитической в области G и не равной тождественно постоянной, не может иметь максимума ни в одной точке области.

Из принципа максимума модуля вытекает важный для нас принцип минимума модуля.

Модуль аналитической функции $f(z)\neq\text{const}$ не может иметь минимума ни в одной точке области, не являющейся нулем функции $f(z)$.

В самом деле, если $f(z_0)\neq 0$, то в силу непрерывности $f(z)$ неравенство $f(z)\neq 0$ имеет место в некоторой окрестности U точки z_0, принадлежащей G. Следовательно, в U функция $\varphi(z)=1/f(z)$ является аналитической и не равной тождественно постоянной. Поэтому модуль $\varphi(z)$ не может иметь максимума в точке z_0. Но тогда $|f(z)|$ не имеет минимума в z_0.

Над комплексной плоскостью мысленно сконструируем поверхность $F(z)=|f(z)|$, называемую аналитическим ландшафтом $f(z)$. Далее, из совпадения нулей $f(z)$ и $F(z)$ и того, что $F(z)\geqslant 0$, вытекает, что нули $f(z)$ соответствуют наиболее «глубоким впадинам» аналитического ландшафта. Но из принципа минимума модуля функции следует, что в аналитическом ландшафте других впадин просто нет. Таким образом, двигаясь по ландшафту от любой его точки «вниз» к комплексной плоскости, будем приближаться к одному из корней $f(z)$. На подобном движении и основан предлагаемый ниже алгоритм «обруча» отыскания корней полиномов. Этот алгоритм следует отнести к разновидности многочисленных методов спуска при

определении минимума функции многих переменных. Правда, здесь для $F(x, y)=|f(z)|$ несколько своеобразно организуется спуск, а его направления и даже их количество заранее не определены. Фактически они являются параметрами алгоритма и должны заново фиксироваться при каждом его выполнении. Варьирование направлений спуска дает возможность получать различные минимизирующие последовательности для каждого конкретного корня $f(z)$. Иногда это позволяет проводить уточняющий анализ полученных приближений к корням.

Приступим к описанию алгоритма.

1. Зафиксируем следующие величины.
   а) начальный угол α_0 $(0\leqslant\alpha_0<2\pi)$;
   б) шаг для угла h $(0<h\leqslant 2\pi/3)$;
   в) «точность» вычислений E $(E>0)$;
   г) количество направлений k $(k=\mathrm{INT}\,(2\pi/h))$.

2. Пусть $E_0=(a_0,\ b_0)$ — точка комплексной плоскости, $R>0$ и $O\,(E_0,\ R)$ — окружность с центром в E_0 и радиуса R. Будем называть $O\,(E_0,\ R)$ «обручем». Исходное положение «обруча» и его размер определим числами: $a_0=b_0=0$, $R=1$.

3. На «обруче» $O\,(E_0,\ R)$ зафиксируем k точек (см. рис. 41)

$$E_j=(a_j,\ b_j)\,(j=\overline{1,\ k}),$$

где

$$a_j=a_0+R\cdot\cos(\alpha_0+(j-1)\,h) \qquad b_j=b_0+R\cdot\sin(\alpha_0+(j-1)\,h).$$

4. Вычислим в точках E_j $(j=\overline{0,\ k})$ значения модуля $F(x, y)$ многочлена $f(z)$. Получим числа F_j $(j=\overline{0,\ k})$.

5. Находим наименьший индекс s такой, что $F_s\leqslant F_j$ $(j=\overline{0,\ k})$. Если $s=0$, то радиус «обруча» уменьшаем вдвое $(R=R/2)$. В противном случае центр «обруча» перемещаем в точку E_s $(E_0=E_s)$, а его радиус увеличиваем в два раза $(R=2\cdot R)$.

6. Если $R>E$, то переходим к шагу 3.

7. Если для $z_0=a_0+ib_0$ $|E(z_0)|\leqslant\varepsilon$, то объявляем $(a_0,\ b_0)$ корнем $f(z)$ и организуем «выщип» корня z_0: $f(z)=F(z)/(z-z_0)$

При $R<E$ и $|F(z_0)|>E$ (например, не можем спуститься из седловой точки) считаем вычисления завершившимися неудачно. Иными словами, при данных α_0 и h $f(z)$ не входит в область определения алгоритма. В этом случае счет можно повторить с другими значениями α_0 и h.

8. Если степень $f(z)$ больше 1, то переходим к шагу 2.

9. Решаем линейное уравнение $f(z)=0$ и находим последний из корней. Вычисления прекращаем.

В дополнение к приведенному алгоритму заметим, что вычисление значений $F(x, y)$ и выделение найденного корня целесообразно проводить по схеме Горнера так, как это реализовано в предыдущей программе. Далее, шаг 9 предполагает непосредственное решение последнего линейного уравнения $f(z)=0$. Если оно выглядит так:

$$(A_0+iB_0)(x+iy)+(A_1+iB_1)=0,$$

то ясно, что

$$x=-(A_0A_1+B_0B_1)\,/\,(A_0^2+B_0^2) \qquad y=(A_1B_0-A_0B_1)\,/\,(A_0^2+B_0^2).$$

И еще один вопрос, в сущности, остался открытым. Какие значения выбирать для α_0 и h? Относительно α_0, не зная корней $f(z)$, по-видимому, ничего определенного сказать нельзя. Что касается h, то здесь нужно исходить из следующих соображений. С одной стороны, хотелось бы перемещаться в сторону минимума $F(x, y)$ по более «правильному» направлению. И это требует брать больше точек на обруче, то есть задавать h небольшим. С другой стороны, при малых h на шагах 3 и 4 алгоритма придется выполнять большой объем вычислений. А это, конечно, плохо. Как же поступать? Исходя из анализа решений тестовых задач, можно рекомендовать пары $(\alpha_0,\ h)$, в которых компоненты берутся из множеств:

$$\alpha_0\in\{0°,\ 30°,\ 45°,\ 60°,\ 90°\} \qquad h\in\{120°,\ 90°,\ 72°,\ 60°\}.$$

Описанный алгоритм реализуется программой примера 4. При этом в случае «плохого» исхода: $R<E$, $|F(z_0)|>E$ — значения α_0 и h подправляются автоматически.

Пример 4. Программа нахождения корней многочленов с комплексными или действительными коэффициентами методом «обруча».

```
10 '    КОРНИ МНОГОЧЛЕНОВ
20 '  (метод 'обруча' для модуля многочлена)
30 '=================================ВВОД
```

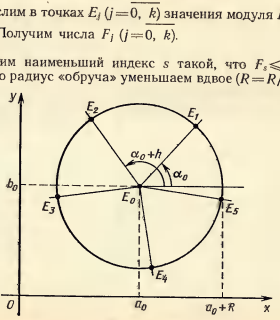


Рис. 41

```
40 SCREEN 0:COLOR 15,4:KEY OFF:WIDTH 39
50 A$=STRING$(38,"-"):D$=STRING$(38,"=")
60 B$="#####.##### +###.#####^I"
70 PRINT "   КОРНИ  МНОГОЧЛЕНА":PRINT A$
80 INPUT "степень многочлена      ";N:M=N
90 DIM A(N),B(N):PRINT
100 PRINT "вещественные части коэффициентов:
110 FOR S=0 TO N
120 PRINT "A(";S;") = ";:INPUT A(S)
130 NEXT:PRINT
140 INPUT "коэф. вещественны (1-ДА,2-НЕТ)";W
150 IF W=1 THEN 200
160 PRINT "мнимые части коэффициентов:"
170 PRINT:FOR S=0 TO N
180 PRINT "B(";S;") = ";:INPUT B(S)
190 NEXT
200 PRINT:INPUT "'точность' вычислений   ";E
210 PRINT A$:PRINT "ДЛЯ ТОЧЕК НА 'ОБРУЧЕ':"
220 INPUT "начальный угол (градусы)";U
230 INPUT "шаг для угла    (1 - 120)";H
240 T0=ATN(1)/45
250 K=360\H:DIM SI(K),CO(K):U=U*T0:H=H*T0
260 FOR S=1 TO K
270 CO(S)=COS(U):SI(S)=SIN(U):U=U+H
280 NEXT:CLS
290 LOCATE 8,3:PRINT "проводятся вычисления"
300 TIME=0:PRINT A$:'==================ЯДРО
310 R=1:X=0:Y=0:F0=1E+50:GOSUB 460:'---обруч
320 A=X:B=Y
330 FOR S=1 TO K
340 X=A+R*CO(S):Y=B+R*SI(S):GOSUB 460
```



```
350 NEXT
360 X=O1:Y=O2:IF X<>A THEN R=R+R:GOTO 320
370 R=R*.5:IF R>E THEN 320
380 IF F0>E THEN U=U+11*T0:GOTO 260
390 N=N-1:GOSUB 560:GOSUB 520
400 IF N>1 THEN 310
410 N=0:D=A(0)^2+B(0)^2:'-------------досчет
420 X=(-A(0)*A(1)-B(0)*B(1))/D
430 Y=(A(1)*B(0)-A(0)*B(1))/D
440 GOSUB 560:GOTO 580
450 '----------------------модуль многочлена
460 T=T+1:D1=A(0):D2=B(0)
470 FOR J=1 TO N
480 L=D1*X-D2*Y+A(J):D2=D2*X+D1*Y+B(J):D1=L
490 NEXT
500 F=D1^2+D2^2:IF F<F0 THEN F0=F:O1=X:O2=Y
510 RETURN
520 FOR J=1 TO N:L=J-1:'--------'выщип' корня
530 A(J)=A(J)+A(L)*X-B(L)*Y
540 B(J)=B(J)+B(L)*X+A(L)*Y
550 NEXT:RETURN:'======================ВЫВОД
560 PRINT "КОРЕНЬ #";M-N;":";
570 PRINTUSING B$;X,Y:RETURN
580 PRINT D$
590 PRINT"общее количество вычислений F =";T
600 PRINT:PRINT "время счета =";TIME/50;"сек
610 END:'============имя программы "КОРНИ-2"
```

Пример 5. Используя программу примера 4, найти все нули многочлена $f(z)=z^3-2z^2+(2+i)z-(i+1)$.

Диалог и результаты счета в данном случае выглядят так.

```
К О Р Н И   М Н О Г О Ч Л Е Н А
-------------------------------------------
   степень многочлена      ? 3
коэф. вещественны (1-ДА, 2-НЕТ) ? 2
вещественные части коэффициентов:
A( 0 ) = ? 1      A( 2 ) = ? 2
A( 1 ) = ? -2     A( 3 ) = ? -1
мнимые части коэффициентов:
B( 0 ) = ? 0      B( 2 ) = ? 1
B( 1 ) = ? 0      B( 3 ) = ? -1
'точность' вычислений    ? 0.00001
-------------------------------------------
ДЛЯ ТОЧЕК НА 'ОБРУЧЕ':
начальный угол (градусы)? 0
шаг для угла    (1 - 120)? 90
        проводятся вычисления
-------------------------------------------
КОРЕНЬ # 1 :    1.00000    +0.00000*I
КОРЕНЬ # 2 :   -0.00000    +1.00000*I
КОРЕНЬ # 3 :    1.00000    -1.00000*I
===========================================
общее количество вычислений F = 154
время счета = 39.2 сек
```

Пример 6. Используя программу примера 4, найти все нули многочлена

$$f(x)=x^6-25x^5+133x^4-789x^3+2780x^2-4100x+2000=$$
$$=(x-20)(x^2+25)(x-1)(x+2)^2.$$

Полный диалог и результаты счета выглядят так

```
К О Р Н И   М Н О Г О Ч Л Е Н А
-------------------------------------------
степень многочлена      ? 6
коэф. вещественны (1-ДА, 2-НЕТ) ? 1
вещественные части коэффициентов:
A( 0 ) = ? 1      A( 4 ) = ? 2780
A( 1 ) = ? -25    A( 5 ) = ? -4100
A( 2 ) = ? 133    A( 6 ) = ? 2000
A( 3 ) = ? -789
'точность' вычислений    ? 0.00001
-------------------------------------------
ДЛЯ ТОЧЕК НА 'ОБРУЧЕ':
начальный угол (градусы)? 60
шаг для угла    (1 - 120)? 120
        проводятся вычисления
-------------------------------------------
КОРЕНЬ # 1 :    2.00000    +0.00000*I
КОРЕНЬ # 2 :    1.00000    +0.00000*I
КОРЕНЬ # 3 :    2.00000    +0.00000*I
КОРЕНЬ # 4 :   -0.00000    +5.00000*I
КОРЕНЬ # 5 :    0.00000    -5.00000*I
КОРЕНЬ # 6 :   20.00000    +0.00000*I
===========================================
общее количество вычислений F = 512
время счета = 181.56 сек
```

В заключение хотелось бы сказать несколько слов о том, что в некоторых случаях задача нахождения корней многочлена оказывается весьма чувствительной к изменению его коэффициентов.

Например, полиномы $f(z)=(z-3)^{14}$ и $g(z)=(z-3)^{14}-10^{-14}$ отличаются только свободными членами на величину 10^{-14}. У первого многочлена один вещественный корень $z=3$ четырнадцатой кратности, а у второго многочлена 14 однократных корней

$$z_k=3+10^{-1}\left(\cos\frac{k\pi}{7}+i\sin\frac{k\pi}{7}\right)\ (k=0,\ 1,\ \ldots,\ 13).$$

Подобную неустойчивость можно наблюдать не только при кратных или почти кратных корнях. Это обстоятельство заставляет с некоторой осторожностью относиться к приближенным значениям корней, полученным тем или иным способом, и требует их тщательного анализа.

Приложение 1. Коды управления принтером [Ф = CHR$ (27)]

Управляющие коды	Название и (или) выполняемые действия
1 Ф; "P" или Ф; "N"	Шаг печати "щщеро" — 10 знаков на дюйм. Шаг задан по умолчанию
2 Ф; "E"	Шаг печати "Элите" — 12 знаков на дюйм
3 Ф; "Q"	"Уплотненный" шаг печати — 17 знаков на дюйм
4 CHR $ (14)	Удвоение действующего основного шага печати
5 CHR $ (15)	Отмена удвоения основного шага печати
6 Ф; "4"	Шрифт "Италик". Включение режима наклонной печати
7 Ф; "5"	Отмена режима наклонной печати Ф; "4"
8 Ф; "G"	Полужирный шрифт. Включение режима печати двойным ударом со смещением изображения вниз
9 Ф; "H"	Отмена режима Ф; "G"
10 Ф; "I"	Полужирный шрифт. Включение режима печати двойным ударом со смещением изображения вправо
11 Ф; CHR $ (34)	Отмена режима Ф; "I"
12 Ф; "X"	Включение режима печати с подчеркиванием
13 Ф; "Y"	Отмена режима Ф; "X"
14 Ф; "s1"	Включение режима верхнего индексирования
15 Ф; "s2"	Включение режима нижнего индексирования
16 Ф; "s0"	Отмена режимов индексирования
17 Ф; "A"	Установка высоты строки в 1/6 дюйма
18 Ф; "B"	Установка высоты строки в 1/9 дюйма
19 Ф; "T"; "n"	Установка высоты строки в n/144 дюйма (n=01, 02, ..., 99)
20 Ф; "L"; "n"	Установка левого поля с позиции n (n=000, 001, ...)
21 Ф; "/"; "n"	Установка правого поля с позиции n (n=000, 001, ...)
22 Ф; "("; "n_1, n_2, ...n_k."	Установка горизонтальных меток табуляции: n_1, n_2, ..., n_k (n_s=001, 002, ...; S=1, 2, ..., k)
23 Ф; ")"; "m_1, m_2, ..., m_P."	Выборочное стирание меток табуляции из позиций: m_1, m_2, ..., m_P (m_s=001, 002, ...; S=1, 2, ..., P)
24 Ф; "2"	Стирание всех меток табуляции
25 CHR $ (9)	Перемещение головки принтера на следующую правую метку
26 Ф; "b"; CHR $ (n)	Перемещение головки принтера на n позиций по строке вправо
27 CHR $ (8)	Возврат головки принтера на 1 позицию назад
28 CHR $ (13)	Возврат головки принтера к началу текущей строки
29 CHR $ (10)	Перемещение головки принтера к началу следующей строки
30 Ф; "a"; CHR $ (n)	Протяжка бумаги на n строк с сохранением позиции головки принтера

Продолжение

Управляющие коды	Название и (или) выполняемые действия
31 CHR $ (12)	Переход к началу следующего листа
32 Ф; "F"; "n"	Перемещение головки принтера на n точечных позиций (n=0000, 0001, ...)
33 Ф; "Rn"; "α"	Повтор n раз символа α (n=001, 002, ...)
34 Ф; ">"	Включение режима однонаправленной печати (медленнее, но качественнее!)
35 Ф; "]"	Отмена режима однонаправленной печати
36 CHR $ (7)	Подача звукового сигнала в течение 1/4 секунды
37 Ф; "p1"	Включение датчика отсутствия бумаги
38 Ф; "p0"	Выключение датчика отсутствия бумаги
39 CHR $ (19)	Перевод принтера в автономный режим, при котором игнорируются все команды с компьютера, кроме CHR $ (17)
40 CHR $ (17)	Вывод принтера из автономного режима
41 CHR $ (24)	«Чистка буфера» принтера
42 Ф; "+"; коды; CHR $ (0)	Макроопределение, представляющее собой совокупность управляющих кодов
43 Ф; "%"	Макрокоманда, активизирующая последнее из выполненных макроопределений
44 Ф; "c1"	Инициализация принтера. Отмена всех введенных CHR $-кодов, кроме новых знаков, созданных пользователем
45 Ф; ".."; CHR $ (n); CHR $ (m_0); CHR $ (m_1); ... ... CHR $ (m_9)	Загрузка нового символа α в память принтера. Здесь: 1) n — код α (n=33, 34, ..., 126); 2) m_0 — характеристика понижения α (m=0, 1); 3) m_s — весовые числа вертикальных рядов α (s=1, 2, ..., 9)
46 Ф; "."	Включение режима вывода новых символов
47 Ф; "&" или Ф; "$"	Отмена режима Ф; "."
48 Ф; "S"; "n" CHR $ (m_{i1}); CHR $ (m_{i2}); ... ... CHR $ (m_{i8})	Вывод 1-й строки изображения я. Здесь: 1) n — количество вертикальных рядов я (n=0001, 0002, ...); 2) m_{is} — весовое число вертикального ряда для строки i (s=1, 2, ..., 9).

Приложение 2. Сообщение о некоторых ошибках.

Сообщение	Код	Перевод и пояснения
1 NEXT without FOR	1	NEXT БЕЗ FOR. Параметры цикла в FOR и NEXT разные
2 Syntax error	2	Синтаксическая ошибка
3 RETURN without GOSUB	3	RETORN БЕЗ GOSUB. Выполнилась команда RETURN без обращения к подпрограмме
4 Out of DATA	4	Недостаток данных в DATA. Не может выполниться очередная команда READ — чтение из блока данных
5 Illegal function call	5	Неправильный вызов функции. Попытка вычислить значение функции при недопустимом аргументе
6 Overflow	6	Переполнение.
7 Out of memory	7	Недостаток памяти.
8 Undefined line number	8	Неопределенный номер строки. Произведено обращение к несуществующей строке
9 Subscript out of range	9	Выход за пределы массива.
10 Redimensioned array	10	Повторное объявление массива. Объявляется массив, место под который уже отведено или по умолчанию, или предыдущей командой DIM
11 Division by zero	11	Деление на нуль.
12 Type mismatch	13	Несоответствие типа. Организуется присваивание переменной одного типа значения другого типа
13 Out of string space	14	Исчерпана строковая область. Командой CLEAR сформируется дополнительное место для строковых значений
14 Device I/O error	19	Ошибка при вводе-выводе.
15 No RESUME	21	Отсутствует команда RESUME.
16 Missing operand	24	Пропуск операнда.
17 Bad file number	52	Неправильный номер файла.
18 File not found	53	Файл не найден. Обращение к файлу, не существующему на данном диске
19 File already open	54	Файл уже открыт.
20 File not OPEN	59	Файл не открыт.
21 File already exists	65	Файл уже существует.
22 Disk full	66	Диск заполнен.
23 Too many files	67	Слишком много файлов.
24 Disk write protected	68	Защита диска от записъ. Переключатель на диске находится в положении, при котором возможно лишь считывание с него
25 Disk I/0 error	69	Ошибка при вводе-выводе
26 Disk offline	70	Дисковод выключен

СПИСОК ЛИТЕРАТУРЫ

1. Пул Л. Работа на персональном компьютере.— М.: Мир, 1986.
2. Моррил Г. Бейсик для ПК ИБМ.— М.: Финансы и статистика, 1987.
3. Уолш Б. Программирование на Бейсике.— М.: Радио и связь, 1987.
4. Биггс Дж. Бейсик: решение производственных задач.— М.: Машиностроение, 1987.
5. Справочное руководство по языку программирования Бейсик для КУВТ «ЯМАХА».— YAMANA corporation, 1986.
6. YAMANA Справочное руководство по языку программирования Бейсик для КУВТ на базе персональных компьютеров «Ямаха MSX-2».— YAMANA corporation, 1988.
7. Эберт К. Компьютеры: Применение в химии.— М.: Мир, 1988.
8. Кетков Ю. Л Диалог на языке Бейсик для мини- и микроЭВМ.— М.: Наука, 1988.
9. Дьяконов В. П. Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ.— М.: Наука, 1987.
10. Информатика и образование.— М.: Педагогика, 1987—1989.— № 1—6.
11. Степанов М. Е. Практические занятия на ПЭВМ Ямаха Минпрос РСФСР. НИИ школ.— М.: 1987.
12. Электронные вычислительные машины: В 8 кн. / Под ред. А. Я. Савельева.— М.: Высшая школа, 1987.
13. Системы автоматизированного проектирования: В 9 кн. / Под ред. И. П. Норенкова.— М.: Высшая школа, 1986.
14. Компьютеры: Справочное руководство: В 3 т. / Под ред. Г. Хелмса. — М.: Мир, 1986.
15. Иодан Э. Структурное проектирование и конструирование программ.— М.: Мир, 1979.
16. Гудман С., Хидитниеми С. Введение в разработку и анализ алгоритмов.— М.: Мир, 1981.
17. Лорин Г. Сортировка и системы сортировки.— М.: Наука, 1983.
18. Вирт Н. Алгоритмы + структуры данных = программы.— М.: Мир, 1985.
19. Мейер Б., Бодуэн К. Методы программирования.— М.: Мир, 1982.— Кн. 1, 2.
20. Кулич И. Л. Математическое программирование.— М.: Высшая школа, 1986.
21. Кофман А. Введение в прикладную комбинаторику.— М.: Физматгиз, 1975.
22. Кофман А., Анри-Добордер А. Методы исследования операций.— М.: Мир, 1977.
23. Моудер Дж., Элмарабн С. Исследование операций.— М.: Мир, 1981.— Т. 1, 2.
24. Заварыкин В. М., Житомирский В. Г., Лапчик М. П. Техника вычислений и алгоритмизация.— М.: Просвещение, 1987.
25. Лапчик М. П. Вычисления. Алгоритмизация. Программирование.— М.: Просвещение, 1988.

ОГЛАВЛЕНИЕ

```
30 A$="T80L804CE-GE- F4E-D G4F4
35 '    C2 E-GB-16.R32B- 05C404B-A- G2X
10 ' ПОДМОСКОВНЫЕ ВЕЧЕРА     В.Соловьев
20 '-----------------------------------
30 A$="T80L804CE-GE- F4E-D G4F4
      C2 E-GB-16.R32B- 05C404B-A- G2X
40 A1$="04A4B405DC04G4
      GD4C G.F16A-4 A-4B-A-G4FE- G4F4
50 PLAY A$:PLAY "05C2"
60 PLAY A1$+"04C2CR32"
70 IF PLAY(1) THEN 70
```