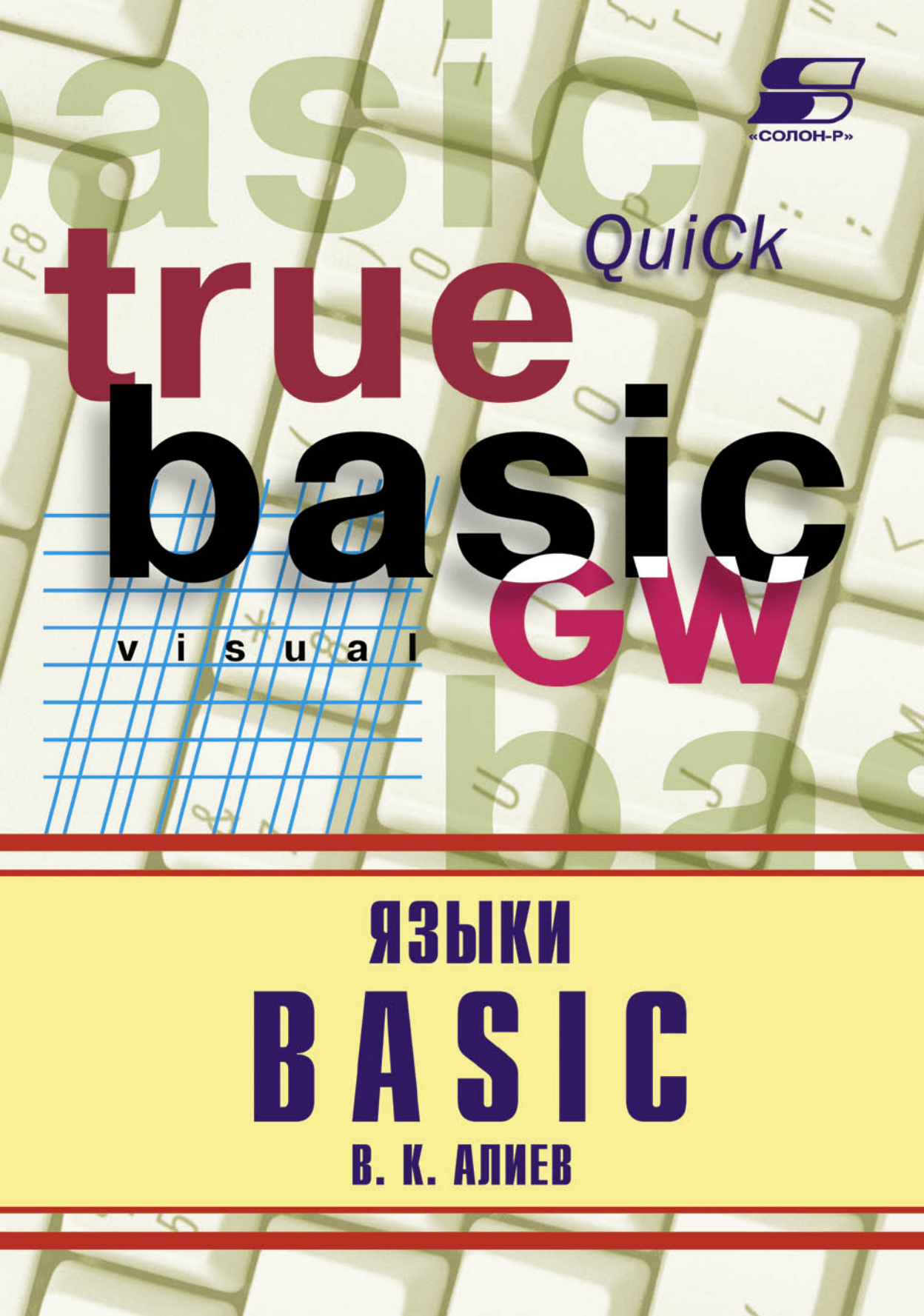

ЯЗЫКИ BASIC

В. К. АЛИЕВ

В.К. АЛИЕВ

Языки Бейсик

Данная книга - самое полное справочное пособие по языку Basic, одновременно является учебником по программированию на языке QBASIC. Особенное внимание уделено практическим примерам программирования - примеры программ (написанные учителями и учащимися старших классов), приведенные в книге, отлажены и проверены в средах Visual Basic и QBasic, поэтому книгу можно использовать как справочник по технике и навыкам программирования.

Предназначена для самого широкого круга читателей-пользователей персонального компьютера, приступающих к изучению или занимающихся программированием - школьников старших классов, учащихся средних специальных учебных заведений, студентов техникумов и вузов, аспирантов, программистов, специализирующихся на VISUAL Basic, независимо от версии.

Данная книга принимала участие в ярмарке педагогических достижений во Владивостоке в октябре 1999 года, где завоевала диплом второй степени в номинации “Учебные пособия и методические разработки”.

Издательство “Солон-Р”

129337, г. Москва, а/я 5

Телефоны:

(499) 254-44-10, (499) 252-36-96

E-mail: Solon.Pub@relcom.ru

Ответственный за выпуск ***С. Иванов***
Макет и верстка ***В. Смирнов***
Обложка ***Е. Жбанов***

ISBN 5—93455—054—3

Глава V. Вывод данных на экран монитора

Помимо приведенных в главе 3.3 операторов PRINT и CLS используются еще несколько операторов вывода данных на экран дисплея.

5.1. Оператор LOCATE

Оператор LOCATE устанавливает курсор в нужное место на экране дисплея и позволяет управлять параметрами самого курсора.

Формат записи оператора LOCATE:

```
LOCATE [номер_строки][,[номер_колонки][{,0,1}]
       [,[начало_курсора][,конец_курсора]]
```

где

Номер_строки — задает место, куда помещается курсор. Значение

Номер_колонки — числового выражения, задающее номер_строки, находится в пределах от 1 до 25; значение числового выражения, задающего номер_колонки задается в пределах от 1 до 40 или 80 в зависимости от установленной длины строки экрана дисплея.

Если третий позиционный параметр равен 0, то курсор будет невидим на экране во время выполнения программы; 1-видим. По умолчанию этот параметр равен 1.

Параметры – **начало_курсора** (верхняя линия) и **конец_курсора** (нижняя линия) задают количество линий сканирующего луча, которые формируют толщину курсора. Значения этих параметров лежат в пределах от 0 до 31. Линии нумеруются сверху вниз от 0 до 13 для алфавитно-цифрового и от 0 до 7 для графического дисплеев.

В результате выполнения оператора LOCATE курсор будет установлен в указанную строку и колонку с заданной видимостью и толщиной. Толщиной курсора можно управлять только в текстовом режиме. Если какой-то операнд опущен, то будет использоваться значение, которое установлено по умолчанию или предыдущим оператором LOCATE.

5.2. Работа с графической информацией

Программные средства языка Бейсик позволяют работать не только с текстовой, но и с графической информацией.

Элементарным изобразительным средством графического экрана является черно-белая или цветная точка разной степени свечения. Координаты точки определяются прямоугольной системой координат и разрешающей способностью экрана дисплея. Разрешающая способность экрана задается оператором SCREEN.

5.2.1. Оператор SCREEN

Оператор SCREEN устанавливает режим и параметры работы экрана дисплея.

Формат записи оператора SCREEN:

```
SCREEN {1,2}[,[cl][,[активная_страница][,отобр_стр]]]
```

где

первый операнд, равный 1, устанавливает графический режим с разрешающей способностью 320 точек по горизонтали и 200 точек по вертикали. Число 2 устанавливает графический режим с разрешающей способностью 640x200. Изображение на экране может быть черно-белым или цветным в зависимости от операнда cl. Если cl=0, то изображение цветное

При разрешающей способности 320x200 горизонтальная строка содержит 320 точек, которые нумеруются слева направо от x=0 до x=319 и вертикально можно расположить 200 точек, которые нумеруются сверху вниз от y=0 до y=199.

Местоположение каждой точки задается ее координатами. Существуют два способа задания координаты точки: абсолютная и относительная.

Абсолютная форма задания координаты точки имеет вид (x,y). Координата точки, которая находится в левом верхнем углу экрана, равна (0,0); координаты точки в правом верхнем углу экрана при SCREEN 1 равны (319,0) и (639,0) при SCREEN 2; координата точки в нижнем левом углу при любом SCREEN равна (0,199).

Относительная форма предполагает задание координат очередной точки относительно последней выведенной точки, осуществляемое с помощью оператора STEP (x1,y1), где x1 и y1 — значения, которые прибавляются к координатам последней выведенной точки.

5.2.2. Оператор PSET

Оператор PSET (Point SET-установить точку) позволяет изобразить точку в указанной позиции экрана дисплея заданного цвета.

Формат записи оператора PSET:

```
PSET (x,y) [,цвет]
```

где

x и y — координаты точек в абсолютной или относительной форме;
цвет — целое число от 1 до 31, которое определяет цвет и насыщенность точки:

0-черный	**1-голубой**	**2-зеленый**
3-бирюзовый	**4-красный**	**5-малиновый**
6-коричневый	**7-белый**	**8-серый**
9-светло-голубой	**10-светло-зеленый**	**11-светло-бирюзовый**
12-светло-красный	**13-светло-малиновый**	**14-желтый**

15-белый насыщенный	**16 и выше-вариации белого**	

5.2.3. Оператор PRESET

Оператор PRESET (Point RESET-изменить яркость точки) позволяет изобразить точку в указанной позиции экрана дисплея заданного цвета.

Формат записи оператора PRESET:

```
PRESET (x,y) [,цвет]
```

где

x и **y** — координаты точек в абсолютной или относительной форме;
цвет — целое число от 1 до 31, которое определяет цвет и насыщенность точки:

Операторы PSET и PRESET отличаются тем, что в первом операторе цвет соответствует цвету переднего плана, а во втором операторе параметр цвет соответствует цвету фона, то есть, если в операторе PRESET цвет не указан, то он гасит точку, изображенную на этом месте оператором PSET.

В случае ошибочного задания координат эти операторы не выполняются. Сообщения об ошибке не выдаются.

5.2.4. Оператор LINE

Оператор LINE позволяет начертить отрезок (линию) или прямоугольник.

Формат записи оператора LINE:

```
LINE [(x1,y1)]-(x2,y2)[,[цвет][,B[F]][,маска]]]
```

где (**x1,y1**) и (**x2,y2**) — координаты начала и конца отрезка, заданные в абсолютной или относительной форме. Если параметры B и F опущены, то в результате выполнения оператора LINE начальная и конечная точки соединяются прямой линией. Если операнд (x1,y1) отсутствует, то в качестве начальной точки используются координаты точки, которая последней участвовала в обработке. После выполнения оператора LINE система запоминает координаты точки (x2,y2).

— параметр **цвет** определяет цвет линии, если до этого оператором SCREEN был установлен соответствующий режим экрана.

— если указан параметр **B**, то система строит прямоугольник. В этом случае параметры (**x1,y1**) и (**x2,y2**) задают координаты противоположных вершин прямоугольника.

— Параметр **F** окрашивает внутреннюю область прямоугольника в указанный цвет. В этом случае параметр цвет на цвет линии не действует.

Пример 5.2.4(1) Построение прямоугольника.

```
'Построение прямоугольника с помощью четырех линий
CLS                          'очистка экрана
SCREEN 12                    'установка параметров экрана
```

```
LINE (200, 50)-(400, 50), 5 'верхняя линия-малиновая
LINE – (400, 100), 1        'правая сторона-голубая
LINE – (200, 100), 2        'нижняя линия-зеленая
LINE – (200, 50), 3         'левая сторона-бирюзовая
LINE (200, 50)-(400, 100), 6'диагональ с правого верхнего
                             угла-коричневая
END
```

Пример 5.2.4(2) Построение вложенных прямоугольников.

```
'Построение вложенных прямоугольников
CLS                                 'очистка экрана
SCREEN 12                           'установка параметров экрана
LINE (200, 155)-(600, 400), 1, BF  'построение голубого
                                    прямоугольника
LINE (240, 165)-(560, 390), 2, BF  'построение зеленого
                                    прямоугольника
LINE (280, 175)-(520, 380), 6, BF  'построение коричневого
                                    'прямоугольника
LINE (320, 185)-(480, 370), 4, BF  'построение красного
                                    прямоугольника
END
```

Пример 5.2.4(3) Построение осей координат.

```
'Построение осей координат для графиков
CLS                                 'очистка экрана
a1: INPUT "введите крайние симметричные значения по оси[1]_
     X(X<0 и X>0)"; x1, x2
IF x1 = 0 OR x2 = 0 THEN GOTO a1    'проверка условий
a2: INPUT "введите крайние симметричные значения по оси_
     У(У<0 и У>0)"; y1, y2
IF y1 = 0 OR y2 = 0 THEN GOTO a2    'проверка условий
SCREEN 12: CLS                      'установка параметров экрана
LINE (0, 199)-(639, 199)            'построение оси X
LINE (319, 0)-(319, 399)            'построение оси У
'ПОСТРОЕНИЕ ЕДИНИЧНЫХ ОТРЕЗКОВ ПО ОСИ X
LINE (0, 197)-(0, 201)              'построение первого отрезка
FOR i = 15 TO 639 STEP 16           'начало цикла формирования
                                     отрезков длиной 4
'построение единичных отрезков по оси X
LINE (i, 197)-(i, 201)
```

[1] Командная строка Бейсика допускает длину 256 символов, что вполне хватает для любого выражения Бейсика, хотя длинные строки менее комфортны на экране. В данной же книге автор постоянно сталкивается с проблемой переноса программных строк, зная по собственному опыту, как порой один лишний символ в ненужном месте делает хорошо отлаженную программу-малопонятным набором инструкций. Именно поэтому все тексты программ автором лично проверены на работоспособность и до сих пор переносы делались только в комментариях. Но все-таки этот момент наступил и автор вынужден идти на некоторое соглашение, а именно: в дальнейшем, как и в данном случае, для переноса программных строк будет применяться символ "_" — подчеркивание, который выбран не случайно, а потому, что именно так переносятся строки в Visual Basic.

```
NEXT i 'конец цикла
'ПОСТРОЕНИЕ ЕДИНИЧНЫХ ОТРЕЗКОВ ПО ОСИ У
LINE (317, 0)-(321, 0)              'построение первого отрезка
FOR i = 9 TO 399 STEP 9             'начало цикла формирования
                                     отрезков с шагом 9
'построение единичных отрезков с длиной 4
LINE (317, i)-(321, i)
NEXT I                              'конец цикла
'ВЫВОД ЗНАЧЕНИЙ КРАЙНИХ ТОЧЕК КООРДИНАТ
LOCATE 12, 1: PRINT x1  'установка курсора в точку x1
LOCATE 12, 78: PRINT x2 'установка курсора в точку x2
LOCATE 24, 42: PRINT y1 'установка курсора в точку y1
LOCATE 2, 42: PRINT y2  'установка курсора в точку y2
'Расчет цены деления
xx = x2 / 10: yy = y2 / 10
LOCATE 26, 52: PRINT "цена деления по оси X — "; xx
LOCATE 27, 52: PRINT "цена деления по оси Y — "; yy
LOCATE 1, 1             'установка курсора в начало экрана
END
```

5.2.5. Оператор CIRCLE

Оператор CIRCLE позволяет получить на экране дисплея эллипс(круг).

Формат записи оператора CIRCLE:

CIRCLE (**x,y**),**r**[,[**цвет**][,[**начало**][,[**конец**][,**сжатие**]]]]

где

(x,y) — координаты центра эллипса, которые можно задавать в абсолютной или относительной форме;

r — любое числовое выражение, значение которого задает большую полуось эллипса.

Значение цвета выбирается из текущей палитры с номером цвета.

0-черный	1-голубой	2-зеленый
3-бирюзовый	4-красный	5-малиновый
6-коричневый	7-белый	8-серый
9-светло-голубой	10-светло-зеленый	11-светло-бирюзовый
12-светло-красный	13-светло-малиновый	14-желтый
15-белый насыщенный	16 и выше-выриации белого	

Операнд **начало** задает начальную точку дуги, **конец** — конечную.

Операнды **начало** и **конец** могут быть любыми числовыми выражениями в диапазоне от -2π до $+2\pi$, где π=3.141593. Эти операнды позволяют получить дугу эллипса.

Операнд **сжатие** задает отношение радиуса, параллельного оси y, к радиусу, параллельному оси x. Оси эллипса всегда параллельны осям координат.

Если значения операндов начало и конец отрицательные, то используются их абсолютные величины и крайние точки дуги эллипса соединяются линией с центром. Если операнды начало и конец опущены, то строится полный эллипс.

Пример 5.2.5(1)

```
'Построение окружностей
SCREEN 12
CLS
CIRCLE (200, 200), 200
CIRCLE (200, 200), 200, 5, , , 3 / 10
CIRCLE (200, 200), 200, 2, , , 4 / 10
CIRCLE (200, 200), 200, 1, , , 5 / 10
END
```

Пример 5.2.5(2)

```
'Построение серии окружностей в цикле
CLS       'Очистка экрана
SCREEN 12 'установка разрешающей способности
          'экрана
FOR r = 1 / 20 TO 10 STEP .1
c = c + 1
IF c > 12 THEN c = 0
CIRCLE (320, 200), 150, 1 + c, , , r 'построение эллипса с цветом
                                     '1+c
NEXT r
PSET (320, 200), 5
END
```

Пример 5.2.5(3)

```
'Построение серии эллипсов в цикле
CLS                    'Очистка экрана
SCREEN 12              'установка разрешающей способности экрана
FOR r = 1 TO 1 / 10 STEP -.1 'начало цикла построение эллипсов
c = c + 1                    'переменная для изменения цвета
IF c > 12 THEN c = 0         'которая изменяется по модулю 12
CIRCLE (320, 200), 150, 1 + c, , , r 'построение эллипса с цветом
                                     '1+c
PAINT (320, 200), 1 + c, 1 + c 'окрашивание эллипса
NEXT r                         'конец цикла
PSET (320, 200), 5             'устанавливаем точку в центре
                               'эллипса
END
```

Глава VI. Операторы работы с экраном монитора

6.1. Преобразование системы координат экрана дисплея

Оператор VIEW(поле зрения) задает прямоугольную область (“окно”) на экране монитора, которое становится текущим экраном. В этом экране будут выполняться все графические построения. Оператор WINDOW устанавливает новую систему координат окна.

6.1.1. Оператор VIEW

Оператор VIEW выделяет на экране монитора “окно”, которое становится новым экраном, доступным для работы. В пределах этого “окна” выполняются все графические построения, а остальные точки экрана становятся недоступными.

Формат записи оператора VIEW:

```
VIEW[[SCREEN][(x1,y1)-(x2,y2)[,[цвет_окна][,[цвет_контура]]]]]
```

где

(x1,y1) и **(x2,y2)** — координаты левого верхнего и правого нижнего углов окна. В операторе можно использовать любые допустимые значения координат x и y, при условии, что x1#x2 и y1#y2. Если все операнды опущены, то “окном” является весь экран.

6.1.2. Оператор WINDOW

Оператор WINDOW устанавливает систему координат окна.

Формат записи оператора WINDOW:

```
WINDOW [[SCREEN][(x1,y1)-(x2,y2)]
```

Если операнд Screen задан, то создается стандартная система координат, то есть **x1** и **y1** задают координату левого верхнего угла окна, а **x2** и **y2** задают координату правого нижнего угла окна. Если же операнд Screen опущен, то создается декартова система координат, где **x1** и **y1** задают координаты левого нижнего угла окна.

При выполнении оператора WINDOW система изменяет координаты x и y независимо от их написания так, чтобы наименьшие значения x и y стали значениями **(x1,y1)**. В операторе можно использовать любые допустимые значения x и y при условии **x1#x2**, **y1#y2**.

Операторы VIEW и WINDOW без операндов отменяют систему координат, определенную ранее оператором WINDOW с операндами.

Пример 6.1.2

```
'программа поиска корней многочлена на числовом отрезке
  CLS
```

```
  DIM min AS DOUBLE 'начало интервала поиска корня
  DIM max AS DOUBLE 'конец интервала поиска корня
  DIM x AS DOUBLE   'переменная
  DIM y1 AS DOUBLE  'функция
  'DIM y2 AS DOUBLE 'функция
  DIM h AS DOUBLE   'точность вычислений
  DIM x1 AS DOUBLE  'корень
SCREEN 9
VIEW (340, 10)-(620, 200), 6, 7 'правое верхнее окно
WINDOW (-14, — 9)-(14, 9) 'устанавливаем систему координат
LINE (-14, 0)-(14, 0), 3  'строим оси
LINE (0, — 9)-(0, 9), 3   'координат
FOR i = — 13 TO 13        'координатные отрезки
LINE (i, — .5)-(i, .5), 4 'по оси x
NEXT i
FOR x = — 10 TO 10 STEP .01 ' построение графика функции
y = x ^ 3 + 5 * x ^ 2 + 3 * x — 8 ' в окне
PSET (x, y), 1
NEXT x
  'поиск корней с указанной точностью
LOCATE 15,1
a: PRINT "введите интервал": INPUT "MIN="; min: INPUT "MAX=";
max
a001: PRINT "введите точность": INPUT "шаг="; h
IF h = 0 THEN
PRINT "ТОЧНОСТЬ СЛИШКОМ ВЕЛИКА ДЛЯ МЕНЯ"
GOTO a001
END IF
    IF min >= max THEN
PRINT "min должен быть меньше max! Повторите ввод!"
GOTO a
     END IF
y1 = min ^ 3 + 5 * min ^ 2 + 3 * min — 8 'вычисляем значение
                                          'ф-ции в начале интервала
y2 = max ^ 3 + 5 * max ^ 2 + 3 * max — 8 'вычисляем значение
                                          'ф-ции в конце интервала
    IF y1 * y2 < 0 THEN
PRINT "в этом интервале существует по крайне мере один корень"
    ELSE
PRINT "в этом интервале корней нет": GOTO a
    END IF
'поиск корня — начало цикла — приближение к корню слева
a01: DO WHILE y1 * y2 < 0
    min = min + h
    y1 = min ^ 3 + 5 * min ^ 2 + 3 * min — 8
    x1 = min
    LOOP ' конец цикла
a1: PRINT "функция="; y1
PRINT "при x="; x1
  INPUT "будете уточнять?(0-нет, точность вычисления — да)"; h
    min = x1
IF h = 0 THEN GOTO m2 'больше не вычисляем
```

```
IF flag% = 0 THEN flag% = 1: GOTO m1 ELSE flag% = 0: GOTO a01
'приближение к корню справа
m1: DO WHILE y1 * y2 > 0
    min = min — h
    y1 = min ^ 3 + 5 * min ^ 2 + 3 * min — 8
    x1 = min
  LOOP: GOTO a1
m2:
  END
```

Глава VII. Управляющие операторы языка Бейсик

До сих пор, рассматривая операторы языка Бейсик и примеры программ, написанных на этом языке, мы обходили молчанием некоторые важные моменты, без понимания которых программирование не существует. Одним из таких основополагающих понятий является ПРОГРАММА для вычислительной машины, основные принципы которой разработала леди Байрон, дочь английского драматурга. Главное, что надо усвоить из этих принципов, которые по существу являются открытиями, это то, что:

1. программа является строго определенной последовательностью команд;

2. команды, из которых состоит программа, условно делятся на выполняемые и управляющие;

3. вычислительная машина выбирает команды программы для выполнения последовательно одну за другой;

4. последовательное выполнение команд может быть нарушено управляющей командой, которая указывает, какая команда выполняется следующей.

Таким образом вводятся понятия выполняемых и управляющих операторов языка программирования.

Выполняемый оператор — это такой оператор, который не нарушает последовательность выполнения операторов программы.

Управляющий оператор — это такой оператор, который может изменить последовательность выполнения операторов программы. И пусть читателя не смущает выражение “который может изменить последовательность выполнения”, которое введено лишь для того, чтобы учесть ситуацию, когда управление передается на оператор, следующий за управляющим.

В языке Бейсик довольно приличный набор управляющих операторов. Но поскольку в задачи данной книги не входит научить писать структурированные программы, а поставлена цель научить пользоваться операторами языка Бейсик, мы рассмотрим следующие управляющие операторы. Наиболее полно операторы Бейсика представлены в главе 11.3. “Ключевые слова Бейсика” и главе XII “Алфавитный указатель операторов, ключевых слов, функций языка Бейсик”.

Табл.7.1.

Оператор	Выполняемое действие
END	Закончить выполнение программы
GOTO	Обеспечить безусловный переход
IF-GOTO	Условный переход
IF-THEN-ELSE	Условный оператор
FOR-TO-NEXT	Арифметический цикл
WHILE-WEND	Итерационный цикл1

DO-LOOP	Безусловный цикл
DO-WHILE(UNTIL)-LOOP	Итерационный цикл2
DO-LOOP-WHILE(UNTIL)	Итерационный цикл3
SELECT-CASE-ENDSELECT	Структура с условием
GOSUB-RETURN	Перейти в(вернуться из) подпрограммы
ON-GOTO	Выбрать безусловный переход
ON-GOSUB	Выбрать подпрограмму
ON-ERROR	Перейти к подпрограмме обработки ошибок
STOP	Временно остановить выполнение программы
DEF	Определить функцию пользователя

7.1. Оператор END

Оператор END заканчивает выполнение программы. При его выполнении закрываются все файлы, которые использовались в программе. Если оператор END отсутствует в программе, то программа заканчивается при выполнении последнего выполняемого оператора.

Формат оператора END

```
END
```

Параметров оператор не имеет. В зависимости от структуры программы оператор END в конце программы можно не указывать. В этом случае выполнение программы закончится, когда не будет операторов для выполнения.

7.2. Оператор GOTO-безусловный переход

Оператор GOTO обеспечивает безусловный переход в любое именованное место программы.

Формат оператора GOTO:

```
GOTO метка_выражения
```

Где

Метка_выражения — имя-переменной типа метки, с помощью которой помечено выражение, которому передается управление. Если управление передано на невыполняемый оператор(REM, DATA и т.п.), то выполнение программы будет продолжено с первого встретившегося выполняемого оператора.

Пример 7.2.1.

Составить программу вычисления выражения y=2^x для значений x<999. Значение x вводится с клавиатуры.

```
CLS 'Очистка экрана монитора
vvod: INPUT "Введите значение X<999"; x 'ввод показателя степени
IF x >= 999 THEN GOTO metka 'проверка введенного числа и
```

```
'если ошибка, то переход на metka
PRINT "2 в степени"; x; "="; 2 ^ x 'печать полученного
результата
'нужно ли еще вычислять?
INPUT "Закончить работу (0-да,#0-нет)"; x%
IF x% <> 0 THEN GOTO vvod 'если не 0, то переход на vvod
END
'обработка ошибки
metka: PRINT "Неверно. Обратите внимание на условие"
GOTO vvod
```

7.3. Оператор IF-THEN-ELSE-условный оператор

Оператор IF-THEN-ELSE (если-то-иначе) обеспечивает принятие двоичного решения.

Формат оператора IF-THEN-ELSE:

```
IF условие1 THEN
    [блок_операторов_1]
[ELSEIF условие2 THEN
    [блок_операторов_2]]...
[ELSE
    [блок_операторов_n]]
END IF
```

где

условие1 Любое выражение Бейсика, которое может быть оценено,
условие2 как истинное(не ноль) или ложное(ноль)

блок_операторов_1 один или несколько операторов в одной
блок_операторов_n или нескольких строках

Другой формат оператора IF-THEN-ELSE:

```
IF условие THEN операторы_1 [ELSE операторы_2]
```

где

условие Любое выражение Бейсика, которое может быть оценено, как истинное(не ноль) или ложное(ноль)

операторы Один или несколько операторов, разделенных двоеточием.

Если отсутствует часть ELSE и условие истинно, то выполняется последовательность операторы_1. После этого управление передается оператору, следующему за оператором IF-THEN-ELSE. Если условие ложно, то оператор IF-THEN-ELSE не выполняется.

Если в операторе IF-THEN-ELSE присутствует часть ELSE и условие истинно, то выполняется последовательность операторов_1. После этого управление передается оператору, следующему за оператором IF-THEN-ELSE. Если условие ложно, то выполняется последовательность операторов_2.

Пример 7.3.1.

```
CLS
a1: INPUT "1 или 2?", i%
IF i% = 1 OR i% = 2 THEN
PRINT "ok"
ELSE
PRINT "вне интервала"
GOTO a1
END IF
```

7.4. Оператор IF-GOTO — оператор условного перехода

Оператор IF-GOTO обеспечивает либо переход GOTO, либо выполнение последовательности операторов, следующих за оператором IF-GOTO.

Формат оператора IF-GOTO:

IF **условие** GOTO **метка_выражения**

где

Условие любое выражение Бейсика, которое может быть оценено, как истинное(не ноль) или ложное(ноль)

Метка_выражения имя переменной типа метки, которой помечено выражение, которому передается управление. Если управление передано на невыполняемый оператор(REM, DATA и т.п.), то выполнение программы будет продолжено с первого встретившегося выполняемого оператора.

Пример 7.4.1.

```
CLS 'очистка экрана монитора
IF a = 0 GOTO b 'условный переход на метку b
PRINT "не работает"
DO: LOOP WHILE INKEY$ = "" 'остановка до нажатия любой клавиши
b: PRINT "работает"
END
```

7.5. Оператор FOR-TO-NEXT-арифметический цикл.

Операторы FOR-TO (для — к) и NEXT(следующий) объединяют в один блок группу операторов, предназначенных для многократного выполнения.

Формат записи оператора:

```
FOR перем._цикла=нач._знач. TO кон._знач. [STEP знач_шага_цикла]
     .............
     тело цикла
     .............
     NEXT [перем._цикла]
```

где

Перем._цикла является числовой переменной.

Нач._знач. и **кон._знач**. могут быть числовой константой или арифметическим выражением.

Знач_шага_цикла может быть как числовой константой, так и числовой переменной, положительной или отрицательной. Если операнд STEP опущен, то значение шага равно 1.

При обращении к оператору FOR-TO сначала вычисляется шаг, начальное и конечное значение. Если при положительном значении шага полученное значение переменной цикла меньше или равно конечному значению, то выполняется последовательность операторов, следующая за оператором FOR-TO(тело цикла). Если полученное значение переменной цикла больше конечного значения, то происходит выход из цикла и управление передается оператору, следующему за NEXT. При отрицательном значении шага значение переменной цикла уменьшается с каждым циклом. Цикл выполняется до тех пор, пока значение переменной цикла не станет меньше конечного значения. Если при входе в блок FOR-TO заданное условие не выполняется, то управление сразу передается оператору, следующему за оператором NEXT. Блоки FOR-TO могут быть вложенными друг в друга.

Пример 7.5.1.

Посчитать число счастливых билетов в катушке автобусных билетов с номерами от 000 000 до 999 999. Счастливым считается билет, у которого сумма первых трех цифр равна сумме последних трех цифр.

```
'Программа подсчета счастливых билетов
FOR i = 0 TO 9 'цикл на первую цифру
 FOR j = 0 TO 9 'цикл на вторую цифру
  FOR k = 0 TO 9 'цикл на третью цифру
   FOR l = 0 TO 9 'цикл на четвертую цифру
    FOR m = 0 TO 9 'цикл на пятую цифру
     FOR n = 0 TO 9 'цикл на шестую цифру
'если совпали, то суммируем
      IF i + j + k = l + m + n THEN s = s + 1
     NEXT n 'конец шестого цикла
    NEXT m 'конец пятого цикла
   NEXT l 'конец четвертого цикла
  NEXT k 'конец третьего цикла
 NEXT j 'конец второго цикла
NEXT i 'конец первого цикла
CLS 'очистка экрана
PRINT "число счастливых билетов="; s 'печать числа счастливых билетов
END 'конец программы
```

По концу работы программа напечатает число 55 252.

Цикл называется арифметическим потому, что до начала цикла переменная цикла должна быть определена, то есть цикл выполняется определенное число раз.

Если в блоке не указан оператор FOR-TO или NEXT, то выдается сообщение:

```
NEXT без FOR
```

или

```
FOR без NEXT
```

Содержание