ВИКТОР ПЕСТРИКОВ, АНДРЕЙ ТЯЖЕВ

QBASIC

НА ПРИМЕРАХ

- ОСНОВНЫЕ ОПЕРАТОРЫ И ФУНКЦИИ
- МЕТОДИКА НАПИСАНИЯ И ОТЛАДКИ ПРОГРАММ
- ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ QBASIC
- БОЛЕЕ 130 ПРИМЕРОВ ПРОГРАММ

bhv®

Виктор Пестриков
Андрей Тяжев

QBASIC
НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.068+800.92Qbasic
ББК 32.973.26-018.1
П28

Пестриков, В. М.

П28 QBASIC на примерах / В. М. Пестриков, А. Т. Тяжев. — СПб.: БХВ-Петербург, 2010. — 304 с.: ил.

ISBN 978-5-9775-0466-9

На многочисленных примерах рассмотрены вопросы программирования на языке QBASIC. Приведено описание основных конструкций алгоритмического языка и показано их использование при решении типовых задач. Для наглядности структуры алгоритма примеры сопровождаются блок-схемами, тексты программ — комментариями. Для закрепления материала подробно рассмотрена разработка программ для игр и создания музыкальных произведений.

Для начинающих программистов

УДК 681.3.068+800.92Qbasic
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Нина Седых*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.08.09.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 24,51.
Тираж 1000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0466-9

Оглавление

40-летию Санкт-Петербургского государственного университета сервиса и экономики (СПбГУСЭ, 1969—2009 гг.) посвящается

Предисловие

Девиз этой книги — "В программирование кратчайшим путем". Это и определяет структуру данного издания, которое начинается с *главы 1 "Создание программы"*, т. е. сразу берем "быка за рога". Поэтому такие разделы, как "Язык программирования BASIC", "Сообщения об ошибке" и "Примеры операторов", обычно располагаемые в начале изданий по языкам программирования, в настоящей книге перенесены в приложения. Их можно изучить и позднее, получив уже опыт создания первых программ на языке BASIC.

Глава 1 книги, посвященная разработке блок-схемы алгоритма, кажется начинающим, на первый взгляд, не особенно-то обязательной, однако это не так. Блок-схема — это фактически созданная программа, записанная языком графов. В этой главе блок-схемы были разработаны авторами, которые постарались показать, как блоки графа переходят в строки программы. Для простых программ их структура интуитивно понятна, и построение блок-схемы кажется излишним. Но есть случаи, когда без блок-схемы разобраться очень трудно.

Первые главы книги составляют, по определению авторов, так называемый "программистский минимум", поскольку, освоив их материал, можно успешно разрабатывать достаточно большие и сложные программы. Весь материал этих глав расположен по принципу эскалации знаний, начальный багаж которых уже дает *глава 1*. Как обычно поступает начинающий, а особенно студент, имея конкретную задачу? Стараясь затратить минимум времени и усилий, он ищет уже готовое решение такой или подобной задачи с тем, чтобы, разобравшись и сделав необходимые изменения, выполнить имеющееся задание. Поэтому в *главе 1* даны семь подробно рассмотренных и прокомментированных примеров на различные темы для того, чтобы дать читателю возможность выбрать что-нибудь подходящее. Если же на какой-нибудь стадии разработки программы, например при организации вывода данных, пользователю будет не хватать знаний, то он сможет обратиться к соответствующей главе (в данном случае это будет *глава 4*) и пополнить свой багаж.

Среди глав книги следует выделить *главы с 3 по 6*. В них рассматриваются операторы, составляющие "костяк" языка BASIC. Действительно, практически в любой программе требуется ввести и вывести данные. Предназначенные для этого операторы приведены в *главе 3* и *4*. Реализовать выбор в программе призваны условные операторы — *глава 5*, а операторы цикла, рассмотренные в *главе 6*, являются мощным оружием любого языка программирования.

Последние две главы книги посвящены вопросам разработки программ для игр и созданию музыкальных произведений.

Среди приложений особое внимание следует обратить на *приложение 3* "Примеры операторов", в котором приведены операторы в рабочем состоянии. Для их использования в собст-

венной программе следует только задать свои параметры. Это приложение специально помещено в конце книги, чтобы было удобнее обращаться к нему.

Авторы старались чаще иллюстрировать материал примерами, причем примерами осмысленными. "...Примеры полезнее правил", — говорил Исаак Ньютон[1]. "...Из глубины разума невозможно извлечь ничего столь значительного и интересного, что можно извлечь из прикладных задач", — вторил ему Л. С. Понтрягин[2]. Трудно не согласиться с этими высказываниями. Поэтому некоторые главы фактически отвечают тенденции — BASIC в примерах и задачах. Представляется, что таким образом проще освоить временами непростой материал. Материал книги прошел апробацию и используется много лет в учебном процессе кафедры "Информатика" Санкт-Петербургского государственного университета сервиса и экономики (СПбГУСЭ) (**www.kafedrainf.ru**) при преподавании курсов "Информатика" (6—10 часов лекций) и "Учебный практикум по вычислительной технике" (36 часов лабораторных работ). Авторы книги будут признательны, если читатели пришлют им свои замечания и предложения по дальнейшему ее совершенствованию (**itss2006@yandex.ru**).

д. т. н., проф. В. М. Пестриков, СПбГУСЭ, Санкт-Петербург

к. т. н., доц. А. Т. Тяжев, СПбГУСЭ, Санкт-Петербург

[1] Сэр Исаак Ньютон (англ. *Sir Isaac Newton*, 04.01.1643—31.03.1727) — великий английский физик, математик и астроном. Автор фундаментального труда "Математические начала натуральной философии" (лат. *"Philosophiae Naturalis Principia Mathematica"*).

[2] Понтрягин Лев Семенович (21.08.1908—03.05.1988) — русский математик, академик АН СССР. Работы школы Понтрягина оказали большое влияние на развитие теории управления и вариационного исчисления во всем мире.

Введение

Выбор языка BASIC (Beginner's All-purpose Symbolic Instruction Code — универсальный код символических инструкций для начинающих) отнюдь не случаен. Его, по праву, можно считать долгожителем: появившись на свет одним из первых среди языков высокого уровня, он пережил как своих "ровесников", так и многих из тех, кто пришел значительно позже. Число версий BASIC, созданных за прошедшие годы для самых разных компьютерных платформ, просто не поддается подсчету. Основной секрет его широкой популярности кроется в удачной комбинации легкости изучения языка и полезности знаний, получаемых в ходе работы с ним.

В настоящее время существуют несколько его версий — GWBASIC, MSX-BASIC, Turbo-BASIC, QuickBASIC. Все версии BASIC схожи друг с другом, и поэтому, изучив одну из них, несложно воспользоваться любой имеющейся версией для решения задач. В данной книге используются QuickBASIC и QBASIC[1].

Осваивая язык BASIC, человек получает универсальную информацию о программировании и языках программирования. Язык программирования BASIC совершенствовался и развивался. Потребность в более быстром, компактном и простом в работе языке программирования привела к появлению Microsoft QuickBASIC.

Использование среды Microsoft QuickBASIC удобно тем, что интерпретатор языка BASIC имеется в любом варианте IBM/PC. Характерные черты языка:

1. Диалоговый режим работы.
2. Нумерация строк.
3. Вещественный и символьный типы данных.
4. Управляющие конструкции.
5. Все переменные являются глобальными.
6. Наличие массивов.

После изучения QuickBASIC переход к использованию профессиональных и высокоуровневых языков становится более простым и естественным делом и позволяет сразу перейти к созданию достаточно сложных приложений. Важным преимуществом является широкая доступность средства QuickBASIC. Объем программы Microsoft QuickBASIC невелик — всего

[1] QBASIC основан на версии QuickBASIC 4.5, но отличается от нее отсутствием компилятора и компоновщика. QBASIC — диалект языка программирования BASIC, разработанный компанией Microsoft.

около 300 Кбайт. Русифицированный вариант программы можно найти в Интернете и скачать, например, по следующим адресам:

- **http://www.filebox.ru/p/qbasic/**
- **http://www.soft.join.com.ua/1549.html** (рис. В1)
- **http://www.klyaksa.net/index.php?htm=htm/download/index.htm**

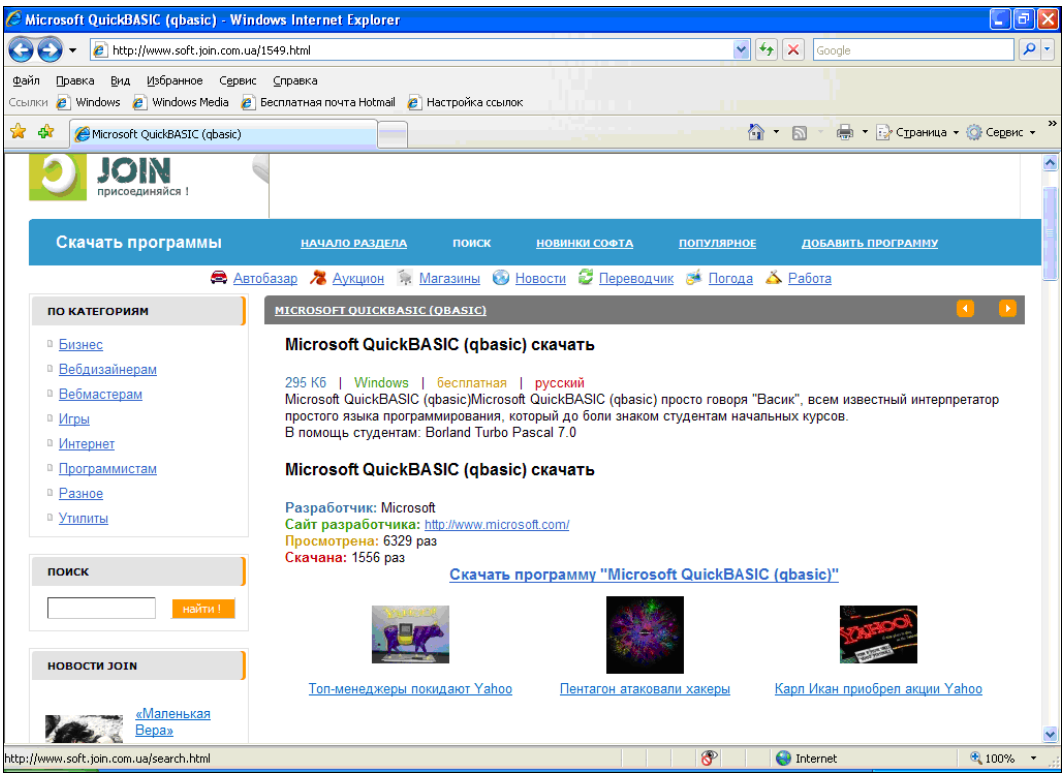


Рис. В1. Сайт **Join.com**, с которого можно скачать программу Microsoft QuickBASIC

Работа в среде программирования QuickBASIC, обычно, изучается на первых курсах технических вузов. В связи с этим всех желающих освежить в памяти написание программ на QuickBASIC отсылаем к электронным учебникам и форумам, имеющимся в Интернете по адресам:

- **http://www.helloworld.ru/texts/comp/lang/qbasic/faq/basic.htm** — BASIC FAQ, наиболее часто задаваемые вопросы в конференции RU.DOS.BASIC;
- **http://www.helloworld.ru/texts/comp/lang/qbasic/book/index.html** — первые шаги в QuickBASIC;
- **http://www.rusedu.info/Article642.html** — сайт "Информационные технологии в образовании" (рис. В2), основы языка QuickBASIC;
- **http://basic113.narod.ru/** — на сайте можно скачать любую версию QuickBASIC, начиная от 1.0 и заканчивая 7.1. Также на сайте есть огромное количество программ, написанных на QuickBASIC, три различных учебника по QuickBASIC и большая справка по всем операторам, функциям и ключевым словам QuickBASIC.

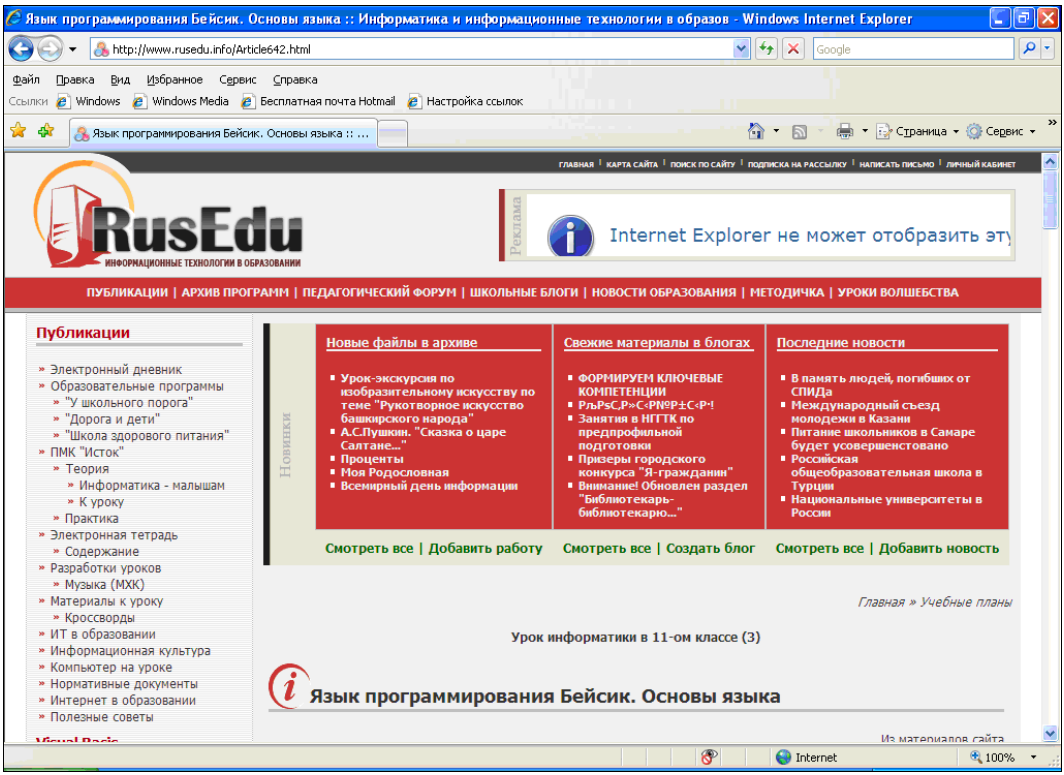
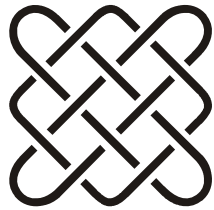


Рис. В2. Сайт "Информационные технологии в образовании"



ГЛАВА 1

Создание программы

1.1. Разработка программ

Для того чтобы приступить к программированию, нужно иметь задачу для программирования. После того как она появится, имеющуюся задачу следует формализовать, т. е. описать ее на языке математики или же используя другие формализованные правила. Способ формализации задачи определяет, как правило, структуру вход-выходных данных, которые могут быть представлены отдельными переменными и константами, массивами и файлами сложной структуры.

Затем необходимо продумать ход решения задачи, т. е. разработать алгоритм ее решения. Алгоритм может быть описан словесно, графом (обычно это блок-схема) и на языке программирования (это и будет программа).

Разработка алгоритма является определяющим этапом в процессе программирования, поскольку задает логическую структуру программы. В соответствии с основной теоремой структурного программирования, доказанной Э. Дейкстрой[1], алгоритм любой сложности можно реализовать, используя только три конструкции: следование (оператор за оператором), повторение (цикл), выбор (альтернатива).

При разработке программы следует иметь в виду, что в укрупненном виде любой алгоритм, а следовательно, и программа, состоит из трех частей:

- ввод данных;
- обработка данных;
- вывод результата.

Язык программирования имеет свой алфавит и зарезервированные слова для команд или операторов, а также служебных инструкций. Зарезервированные слова нельзя использовать в качестве меток или имен переменных, констант и процедур. Кроме команд или операторов, в программах используются числовые и символьные переменные, а также константы. В табл. 1.1 приведены имеющиеся в языке функции, а в *приложении 3* — примеры операторов.

---

[1] Эдсгер Дейкстра — почетный заведующий кафедрой, созданной компаниями Schlumberger и Centennial на факультете информатики (Computer Sciences) университета штата Техас. Он является выдающимся представителем научного направления, которое принято называть теоретическим программированием. Вся его научная деятельность была направлена на разработку средств и методов для создания "правильных" программ, т. е. таких программ, корректность которых может быть доказана формальными методами.

Таблица 1.1. *Операции и функции в BASIC*

Операции, функции	Вид в BASIC
Сложение	+
Вычитание	-
Умножение	*
Деление	/
Возведение в степень	^
Целочисленное деление	\
e^x	EXP(x)
$\|x\|$	ABS(x)
sin *x*	SIN(x)
cos *x*	COS(x)
tg *x*	TAN(x)
ln *x*	LOG(x)
lg *x*	LOG(x) / LOG(10)
$\sqrt{x}$	SQR(x)
arctg *x*	ATN(x)
Остаток от деления	i MOD j
Генератор случайных чисел	RND(n)
Равно	=
Больше	>
Меньше	<
Не равно	<>
Меньше или равно	<=
Больше или равно	>=

Примечание

1. Согласно приоритету операций в BASIC сначала выполняется возведение в степень, затем умножение и деление, после сложение, вычитание. Операции, размещенные в одной строке и имеющие равный приоритет, выполняются последовательно слева направо.
2. BASIC вычисляет выражение в скобках в первую очередь, даже если операции в скобках более низкого приоритета, чем вне скобок. Сначала вычисляются внутренние скобки, затем наружные.
3. Обратные функции вычисляются через имеющиеся встроенные:

   секанс: `SEC(x) = 1 / COS(x);`

   косеканс: `COSEC(x) = 1 / SIN(x);`

   арксинус: `ARCSIN(x) = ATN(x / SQR(1 - x * x));`

   арккотангенс: `ARCCTN(x) = 1.570796 - ATN(x);`

   арккосинус: `ARCCOS(x) = 1.570796 - ATN(x / SQR(1 - x * x)).`

Рассмотрим процесс разработки программы на несложном экономическом примере.

Пример 1.1. Экономическая задача

Требуется разработать программу расчета времени накопления заданной суммы средств *Kk* от начальной суммы вклада *Kn*, если процентная ставка равна *p*, при условии начисления простых и сложных ежегодных процентов. Сравнить эти варианты.

Задача формализована, поскольку имеются формулы для вычисления суммы при заданных видах процентов:

$S = Kn \cdot (1 + t \cdot p/100)$ — для простых процентов,

$S = Kn \cdot (1 + p/100)^t$ — для сложных процентов,

где *Kn* — начальная сумма вклада; *p* — годовая процентная ставка; *t* — срок вклада.

Для полной ясности скажем, что простые проценты — это проценты, ежегодно (в нашем случае) начисляемые лишь на начальную сумму вклада, а при сложных процентах происходит капитализация процентов, т. е. проценты начисляются и на проценты, прибавляемые к начальной сумме вклада. Понятно, что при начислении сложных процентов заданная сумма накопится быстрее.

Алгоритм решения задачи в виде блок-схемы приведен на рис. 1.1. Процесс построения блок-схем подробно рассмотрен в *разд. 1.2*. Если располагать текст программы и соответствующую блок-схему рядом (т. е. параллельно), то можно проследить, как блоки блок-схемы переходят в блоки или строки разрабатываемой программы.

Программа 1.1

```
'1ekonom.bas     Экономическая задача
10  CLS
20  PRINT "Пример 1.1"
30  PRINT "Расчет времени накопления заданной суммы средств"
40  PRINT "при условии начисления простых или сложных ежегодных"
50  PRINT "процентов. Сравнение этих двух вариантов.": PRINT
60  PRINT "Простые проценты — 1"
70  PRINT "Сложные проценты — 2"
80  PRINT "Сравнение вариантов — 3"
90  INPUT "Выбор — ", vib
100 IF vib = 1 OR vib = 2 OR vib = 3 THEN 120 ELSE
110 PRINT "Такого варианта нет": SLEEP 2: GOTO 10
120 PRINT: INPUT "Начальная сумма вклада в руб. = ", Kn
130 INPUT "Ставка годовых % = ", p
140 INPUT "Сумма накопления в руб. = ", Kk
'==1===== Блок выбора ============================================
150 IF vib = 1 THEN 170
160 IF vib = 2 THEN 240
'==2===== Блок вычисления простых процентов =======================
170 DO
180 t1 = t1 + 1
190 S1 = Kn * (1 + t1 * p / 100)
200 IF S1 >= Kk THEN EXIT DO
210 LOOP
220 t = t1: PRINT
230 IF vib = 3 THEN  ELSE 330
```

```
'==3===== Блок вычисления сложных процентов ======================
240 DO
250 t2 = t2 + 1
260 S2 = Kn * (1 + p / 100) ^ t2
270 IF S2 >= Kk THEN EXIT DO
280 LOOP
290 t = t2: PRINT
300 IF vib = 3 THEN  ELSE 330
'==4===== Блок сравнения вариантов ================================
310 t = t1 - t2
320 PRINT "Разница во времени накопления - ", t: GOTO 340
'==5===== Блок вывода результата ==================================
330 PRINT "Время накопления заданной суммы"; t;
340 IF t < 5 THEN  ELSE PRINT "лет"
350 IF t = 1 THEN  PRINT "год" ELSE IF t < 5 THEN PRINT "года"
'================================================================
360 END
```

Программу рекомендуется начинать со строки комментариев, в которой в качестве названия программы использовать название файла, в который записана эта программа. Так в данном случае это будет — 1ekonom.bas. Затем в этой же строке желательно дать небольшой комментарий с описанием программы.

Как правило, программу в BASIC начинают с оператора CLS — очистка экрана, чтобы на экране была только информация, относящаяся к этой программе, а посторонняя информация не мешала бы работе.

Теперь о нумерации строк программы. Она обязательна только для тех строк, на которые есть ссылки в операторах GOTO, GOSUB, IF...THEN и т. д. Нумерация в программе удобна в том случае, если необходимо комментировать эту программу. В BASIC принято нумеровать строки с шагом 10 для того, чтобы при внесении изменений в текст программы нумеровать новые строки в промежутках существующей нумерации, например между строками с номерами 10 и 20 можно вставить строки с номерами 11, 12, 13 и т. д. При этом номера других линий останутся без изменений, а значит не нужно корректировать ссылки в указанных операторах.

После оператора CLS желательно с помощью оператора PRINT вывести на экран название программы, а, может быть, и краткую формулировку поставленной задачи. В программе 1.1 заголовок выводится оператором PRINT в строке 20, а краткая формулировка дана в строках 30—50. Операторы PRINT в строках 60—80 выводят на экран своеобразное меню, подсказывающее варианты выбора. Таким образом, блок 1 блок-схемы на рис. 1.1 отражен строками 60—90 программы 1.1. Строки 100—110 программы осуществляют анализ сделанного выбора. В литературе часто это называют "защитой от дурака". Авторы не согласны с такой формулировкой и предлагают называть это "блокировкой любознательного", слишком любознательного.

Блок 2 блок-схемы реализуется с помощью операторов INPUT в строках 120—140 программы. Оператор приостанавливает работу программы и ожидает ввода данных с клавиатуры. Ввод заканчивается нажатием клавиши <Enter>. Переменные в операторах INPUT в строках 120—140 являются числовыми, поэтому программа ожидает ввода чисел. Попытка ввода текста вызовет появление сообщения "Ввод сначала". Выражение в кавычках при операторе INPUT обычно называют "подсказкой" или "приглашением к вводу", и оно является весьма желательной вещью для правильного оформления программы. Подробнее оператор INPUT будет рассмотрен в *разд. 3.2*.

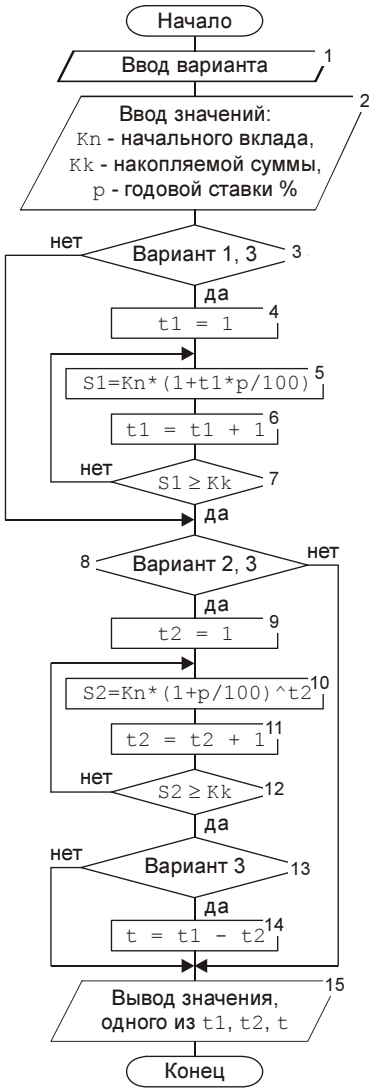


Рис. 1.1. Блок-схема экономической задачи

Ветвление, задаваемое блоками 3, 8, 13 блок-схемы, осуществляется посредством условного оператора `IF...THEN` в строках 150—160, 230, 300 программы 1.1. Между ключевыми словами `IF` и `THEN` условного оператора помещается условие. Если условие выполняется, то осуществляется переход по метке, находящейся после ключевого слова `THEN`, а если после слова `THEN` находятся операторы, то они выполняются. Подробнее условный оператор `IF...THEN` будет рассмотрен в *разд. 5.1*.

Программа 1.1 содержит два цикла `DO...LOOP` по годам (`t1 = t1 + 1` и `t2 = t2 + 1`), первый из которых создан для вычисления времени накопления заданной суммы при простых процентах (цикл в строках 170—210, соответствующая формула вычисляется в строке 190), а второй — для вычисления времени накопления заданной суммы при сложных процентах (цикл в строках 240—280, формула в строке 260). Отметим, что циклический оператор

DO...LOOP предназначен для организации цикла с заранее неизвестным числом повторений, а завершение цикла происходит по заданному условию. Подробнее о циклическом операторе DO...LOOP читайте в *разд. 6.4*.

В строках 180 и 250 на каждом шаге циклов происходит увеличение на 1 счетчика циклов по годам (от 1 до искомого значения). Выход из цикла организован одинаково в обоих вариантах в строках 200 и 270 с помощью условного оператора IF...THEN, в который поставлен оператор EXIT DO. Условие выхода из циклов — накопление заданной суммы (S1 >= Kk и S2 >= Kk). Строки 220 и 290 подготавливают к выводу на экран найденного время накопления заданной суммы, осуществляемого посредством оператора PRINT в строке 330. Операторы PRINT в строках 220 и 290 задают пустую строку, отделяющую вывод времени накопления от предыдущего текста. Условные операторы IF...THEN в строках 340—350 предназначены для согласования величины значения t при выводе с последующим пояснением ("год", "года", "лет").

Теперь пройдемся по ветвлениям программы 1.1.

Пусть выбран первый вариант — простые проценты. Тогда условный оператор IF...THEN в строке 150 задает переход на строку 170. Начинает работать цикл DO...LOOP в строках 170—210 по вычислению времени накопления заданной суммы в случае простых процентов. По завершении работы этого цикла условный оператор IF...THEN в строке 230 задает по ELSE переход на линию 330, где находится выводящий на экран оператор PRINT.

Теперь пусть выбран второй вариант — сложные проценты. Тогда условный оператор IF...THEN в строке 160 задает переход на строку 240. Начинает работать второй цикл DO...LOOP в строках 240—280 по вычислению времени накопления заданной суммы в случае сложных процентов. После завершения вычисления времени накопления условный оператор IF...THEN в строке 300 задает переход на линию 330 для вывода результата на экран.

И, наконец, последний — третий вариант. В этом случае условные операторы IF...THEN в строках 150—160 пропускают в цикл DO...LOOP в строках 170—210. Затем условный оператор IF...THEN в строке 230 пропускает в цикл DO...LOOP в строках 240—280. После чего условный оператор IF...THEN в строке 300 передает управление на строку 310 вычисления разницы во времени накопления при условии простых и сложных процентов (t = t1 – t2). Результат на экран выводит оператор PRINT в строке 320.

Итак, блоки 4—7 блок-схемы на рис. 1.1 переходят в строки 170—210 программы 1.1, причем блок 5 реализуется строкой 190, блок 7 — строкой 200, а блоки 4, 6 — строкой 180.

```
Пример 1.1
Расчет времени накопления заданной суммы средств
при условии начисления простых или сложных ежегодных
процентов. Сравнение этих двух вариантов.

Простые проценты - 1
Сложные проценты - 2
Сравнение вариантов - 3
Выбор - 1

Начальная сумма вклада в руб. = 1000
Ставка годовых % = 10
Сумма накопления в руб. = 2000

Время накопления заданной суммы 10 лет
```

Рис. 1.2. Вывод результата вычисления времени накопления заданной суммы при условии простых процентов

Аналогично, блоки 9—12 отображаются в строки 240—280, блоки 9, 11 реализуются строкой 250, блок 10 — строкой 260, а блок 270, определяющий условие выхода из цикла, — строкой 270.

Блок 14 рассматриваемой блок-схемы переходит в строку 310 программы 1.1, а блок 15 реализуется операторами `PRINT` в строках 320—350.

Результат работы программы 1.1 приведен на рис. 1.2.

Пример 1.2. Олимпийские кольца

Требуется разработать программу, рисующую олимпийские кольца.

Этот пример показывает, как можно, используя массивы, применить циклы для обработки нерегулярных данных. В программе, рисующей олимпийские кольца, такими нерегулярными данными является цвет колец: синий — 11, черный — 8, красный — 12, желтый — 14 и зеленый — 10 (номера цветов BASIC смотрите в табл. 8.2 в *разд. 8.1*). Рекомендуется перед разработкой программы предполагаемый рисунок изобразить на бумаге в клетку, что позволит определить ориентировочные координаты опорных точек фигуры, которые будут уточняться в процессе рисования. Подобный эскиз приведен на рис. 1.3.

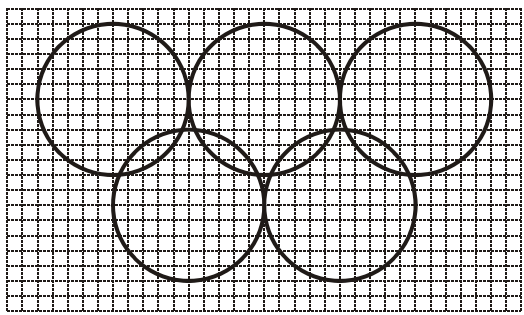


Рис. 1.3. Олимпийские кольца

Блок-схему для этой программы предлагается читателю разработать самостоятельно, и поэтому обратимся непосредственно к программе.

Программа 1.2

```
'OlimpKol.bas    Олимпийские кольца
10  CLS                   'очистка экрана
20  SCREEN 9              'установка экранного режима
30  COLOR 14, 9           'установка экранных цветов (9 — цвет фона)
40  DIM cvet(5)            'объявляется массив для записи номеров цвета
'==1===== Блок рисования колец ====================================
50  xn = 200: xk = 400
60  FOR y = 100 TO 150 STEP 50
70  FOR x = xn TO xk STEP 100
80  DATA 11,8,12,14,10
90  k = k + 1: READ cvet(k)
100 CIRCLE (x, y), 50, cvet(k)
110 CIRCLE (x, y), 49, cvet(k)
```

```
120 CIRCLE (x, y), 48, cvet(k)
130 NEXT x
140 xn = 250: xk = 350
150 NEXT y
'================================================================
160 END
```

Кольцо рисуется операторами CIRCLE в строках 100—120 (3 оператора со смежными значениями радиуса — 50, 49, 48 для создания толщины). Координаты центров колец задаются посредством двойного цикла (внешний в строках 60—150 задает координату y, а внутренний в строках 70—130 задает x). Цвет колец вводят операторы READ и DATA в строках 80—90. Подробнее об этих операторах читайте в *разд. 3.3*. Поскольку в верхнем ряду 3 кольца, а в нижнем — только 2, то и начальные и конечные значения счетчика цикла в строке 70 будут меняться в зависимости от ряда. Для верхнего (3 кольца) они задаются в строке 50, а для нижнего (2 кольца) — в строке 140.

Теперь рассмотрим решение оптимизационной задачи.

Пример 1.3. Оптимизационная задача "Диета"

Фирма занимается составлением диеты, содержащей по крайней мере 20 единиц белков, 30 единиц углеводов, 10 единиц жиров и 40 единиц витаминов. Как дешевле всего достичь соблюдения диеты при указанных в табл. 1.2 ценах (в рублях) на 1 кг (или 1 л) имеющихся пяти продуктов?

Таблица 1.2. *Содержание элементов диеты в пяти продуктах и их цена*

	Хлеб	Соя	Сушеная рыба	Фрукты	Молоко
Белки	2	12	10	1	2
Углеводы	12	0	0	4	3
Жиры	1	8	3	0	4
Витамины	2	2	4	6	2
Цена	**12**	**36**	**32**	**18**	**10**

Блок-схема алгоритма решения этой задачи приведена на рис. 1.4.

Программа 1.3

```
'1Dieta.bas       Диета
10 CLS
20 Cmin = 10000
'==1===== Блок оптимизации ============================================
30 FOR x1 = 0 TO 20
40 LOCATE 1, 1: PRINT "cikl ="; x1
50 FOR x2 = 0 TO 20
60 FOR x3 = 0 TO 10
70 FOR x4 = 0 TO 20
80 FOR x5 = 0 TO 20
90 u1 = 0: u2 = 0: u3 = 0: u4 = 0
```

```
'---- Блок проверки условий ---------------------------------------
100 IF 2 * x1 + 12 * x2 + 10 * x3 + x4 + 2 * x5 >= 20 THEN u1 = 1
110 IF 12 * x1 + 4 * x4 + 3 * x5 >= 30 THEN u2 = 1
120 IF x1 + 8 * x2 + 3 * x3 + 4 * x5 >= 10 THEN u3 = 1
130 IF 2 * x1 + 2 * x2 + 4 * x3 + 6 * x4 + 2 * x5 >= 40 THEN u4 = 1
'------------------------------------------------------------------
140 IF u1 + u2 + u3 + u4 = 4 THEN  ELSE 180
150 c = 12 * x1 + 36 * x2 + 32 * x3 + 18 * x4 + 10 * x5
160 IF c < Cmin THEN  ELSE 180
170 Cmin = c: Xo1 = x1: Xo2 = x2: Xo3 = x3: Xo4 = x4: Xo5 = x5
180 NEXT x5, x4, x3, x2, x1
'==2===== Блок вывода результата ==================================
190 PRINT "Минимальная стоимость диеты"; Cmin; "рублей"
200 PRINT "Оптимальное количество хлеба = "; Xo1; "буханок"
210 PRINT "Оптимальное количество сои = "; Xo2; "кг"
220 PRINT "Оптимальное количество сушеной рыбы = "; Xo3; "кг"
230 PRINT "Оптимальное количество фруктов = "; Xo4; "кг"
240 PRINT "Оптимальное количество молока = "; Xo5; "литров"
'==================================================================
250 END
```

Результат работы программы 1.3 показан на рис. 1.5.

Быстродействие современных компьютеров позволяет решать оптимизационные задачи самым простым образом — путем перебора, осуществляемого в рассматриваемом примере по пяти видам продуктов. Введем следующие обозначения для переменных:

- `x1` — для хлеба;
- `x2` — для сои;
- `x3` — для сушеной рыбы;
- `x4` — для фруктов;
- `x5` — для молока.

Таким образом, для осуществления перебора следует организовать пятерной вложенный цикл (с помощью операторов цикла `FOR...NEXT` в строках 30—180 программы 1.3), в теле которого на каждом шаге проверяются условия по количеству белков (блок 8 блок-схемы на рис. 1.4), углеводов (блок 11), жиров (блок 14) и витаминов (блок 17). Данные для условий берутся из табл. 1.2. Например, для белков: `x1` единиц хлеба дает `2 * x1` единиц белка, `x2` единиц сои дает `12 * x2` единиц белка, `x3` единиц сушеной рыбы дает `10 * x3` единиц белка и т. д. Следовательно, условие по белку будет выглядеть следующим образом:

```
2 * x1 + 12 * x2 + 10 * x3 + x4 + 2 * x5 ≥ 20
```

и реализуется в блоках 8—10 блок-схемы и посредством условного оператора `IF...THEN` в строке 100 программы 1.3.

Аналогично определяются условия по углеводам (блоки 11—13 и строка 110):

```
12 * x1 + 4 * x4 + 3 * x5 ≥ 30,
```

по жирам (блоки 14—16 и строка 120):

```
x1 + 8 * x2 + 3 * x3 + 4 * x5 ≥ 10,
```

по витаминам (блоки 17—19 и строка 130):

```
2 * x1 + 2 * x2 + 4 * x3 + 6 * x4 + 2 * x5 ≥ 40.
```

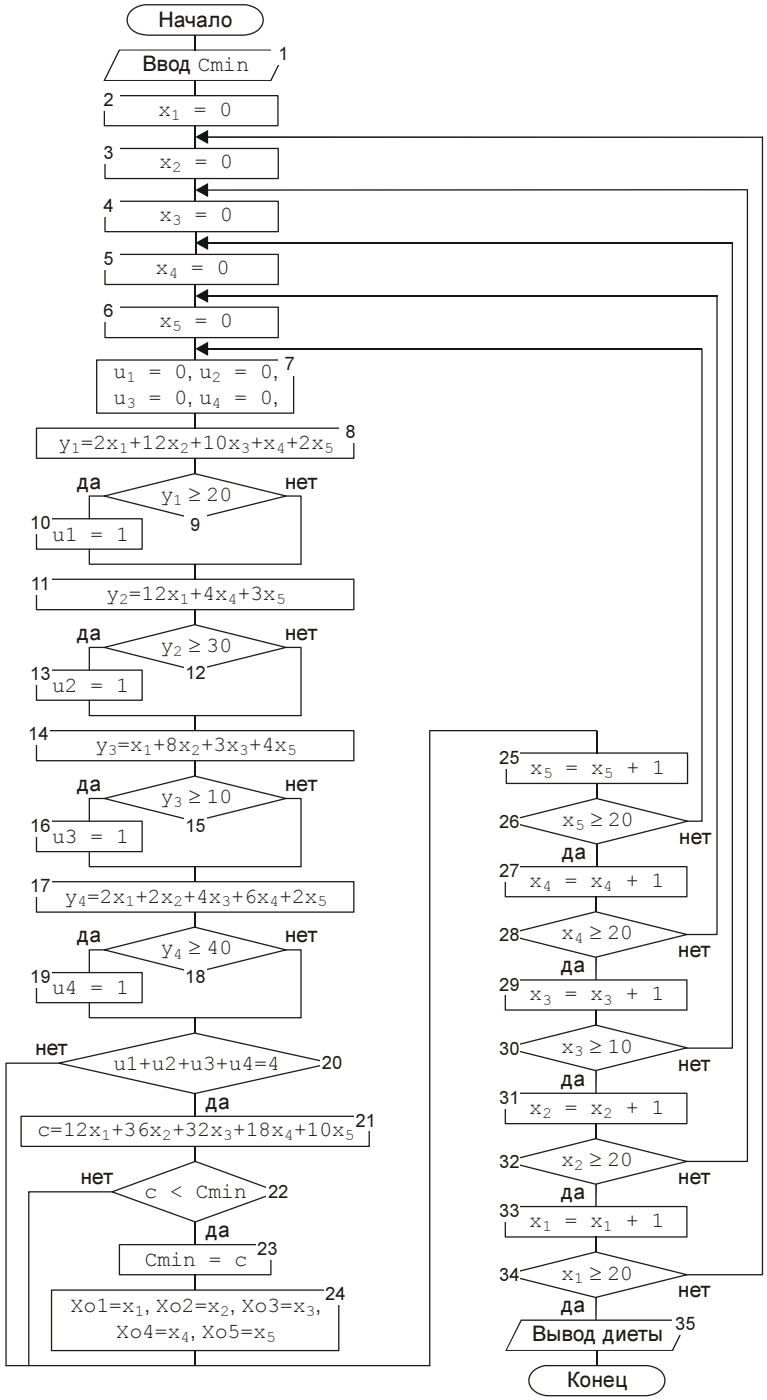


Рис. 1.4. Блок-схема оптимизационной задачи

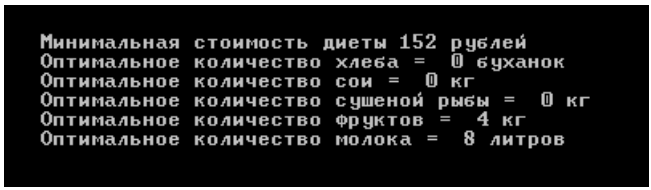


Рис. 1.5. Вывод программы "Диета"

В случае одновременного выполнения всех четырех условий (блок 20 и строка 140, все вспомогательные переменные `u1`—`u4` равны 1, а, значит, их сумма равна 4), вычисляется стоимость набора продуктов (блок 21 и строка 150), после чего эта стоимость сравнивается с предыдущей минимальной стоимостью `Cmin` (блок 22 и строка 160). Если вычисленная стоимость меньше предыдущей минимальной, то она принимается за новую минимальную (блок 23 и строка 170). Описанная процедура повторяется до окончания полного перебора.

Верхние границы перебора определяются по табл. 1.2 по строке "Витамины" для всех продуктов, за исключением фруктов. Для них верхняя граница перебора определяется по строке "белки". Поясним логику выбора на примере хлеба. Условие по витаминам выполняется при `x1 = 20` (`2 * x1 = 40`), и при этом заведомо выполняются соответствующие условия и по остальным показателям — по углеводам `12 * x1 = 240 > 30`, по белкам — `2 * x1 = 40 > 20` и по жирам — `x1 = 20 > 10`.

В строках 190—240 программы 1.3 посредством шести операторов `PRINT` производится вывод на экран состава минимальной по стоимости диеты и ее цена.

Следующий пример на совместное использование графики и звука взят из книги [9].

Пример 1.4. Гамма

Разработать программу, играющую гамму, т. е. воспроизводящую ноты первой октавы вначале в прямом, а затем в обратном порядке. В ходе выполнения программы на экран выводится нотный стан с нотами, причем звучащая нота выделяется другим цветом.

Для решения поставленной задачи разработана программа 1.4, результат работы которой показан на рис. 1.6.

Блок-схема алгоритма решения задачи, поставленной в примере 1.4, приведен на рис. 1.7. Из рисунка видно, что программа должна состоять из четырех циклов:

- цикл для ввода нот — блоки 1—4;
- цикл для рисования нотного стана и нот — блоки 5—9;
- цикл для озвучивания нот в прямом порядке и выделения звучащей ноты другим цветом — блоки 10—14;
- цикл для озвучивания нот в обратном порядке и выделения звучащей ноты другим цветом — блоки 15—19.

В программе 1.4 эти четыре цикла реализуются посредством операторов цикла `FOR...NEXT` в строках 50—120 (внешний цикл) и в строках 90—120 (вложенный цикл), а также условного оператора `SELECT CASE`, являющегося телом вложенного цикла. Оператор `FOR...NEXT` с числом повторений 4, образующий внешний цикл, обеспечивает реализацию всех четырех перечисленных циклов. А оператор цикла `FOR...NEXT` в строках 90—120 с изменяемыми параметрами счетчика цикла конкретизирует указанные циклы.

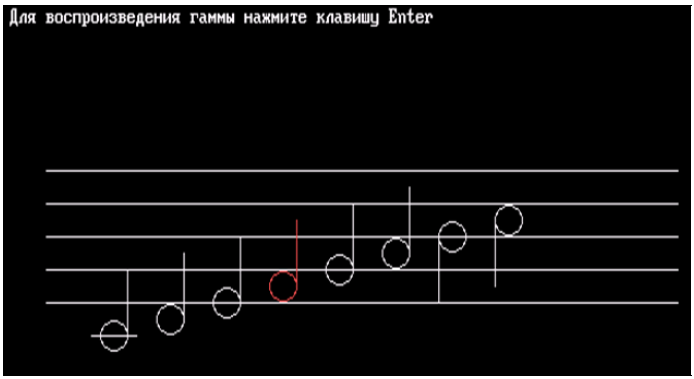


Рис. 1.6. Вывод программы "Гамма"

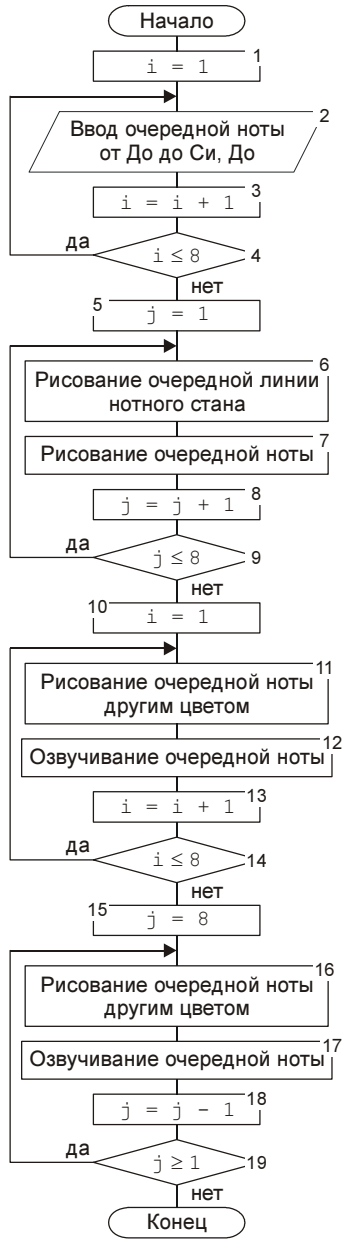


Рис. 1.7. Блок-схема программы "Гамма"

Программа 1.4

```
'1gamma.bas          Гамма
10 CLS                    'очистка экрана
20 SCREEN 9               'установка экранного режима
30 COLOR 8, 15            'установка экранных цветов (15 — цвет фона)
40 DIM nota$(8)           'готовит в памяти место для нот
```

```
'==1===== Блок рисования и исполнения гаммы ======================
50  FOR j = 1 TO 4
60  IF j < 4 THEN in = 1: ik = 8: p = 1 ELSE in = 8: ik = 1: p = -1
70  IF j = 3 THEN  ELSE 90
80  PRINT " Для воспроизведения гаммы нажмите клавишу Enter": SLEEP
90  FOR i = in TO ik STEP p
100 SELECT CASE j
       CASE 1: READ nota$(i)
                    DATA "n1","n2","n3","n4","n5","n6","n7","n8"
       CASE 2: cv = 8
                    GOSUB 200          'рисует нотный стан и ноты
       CASE 3, 4: cv = 12
                    GOSUB 300          'выделяет ноту и играет гамму
                    n$ = nota$(i)
                    PLAY "n0 x" + VARPTR$(n$) + "n0"
                    cv = 8             'cv — задает цвет
                    GOSUB 300
110 END SELECT
120 NEXT i, j
130 GOTO 400
'==2===== Подпрограмма рисования нотного стана ====================
200 IF i = 1 THEN x1 = 80: x2 = 120 ELSE x1 = 40: x2 = 600  'линия для До
210 IF i < 7 THEN LINE (x1, 220 — i * 20)-(x2, 220 — i * 20)
220 GOSUB 300
230 RETURN
'==3===== Подпрограмма рисования нот ==============================
300 CIRCLE (50 + i * 50, 210 — i * 10), 12, cv
310 IF i < 7 THEN b = 1 ELSE b = -1          'b = -1 — хвостик ноты вниз
320 LINE (50 + 50 * i + 12 * b, 210 — i * 10)-STEP(0, -40 * b), cv
330 RETURN
'==================================================================
400 END
```

Действительно, на 1-м шаге внешнего цикла со счетчиком цикла `j` по `CASE 1` условного оператора `SELECT CASE` в строках 100—110 записываются в память "звуковые" команды для оператора `PLAY`, определяющие последующее проигрывание соответствующих нот. На 2-м шаге по `CASE 2` с помощью оператора `GOSUB 200` задается обращение к подпрограмме рисования нотного стана, в которой в свою очередь записано обращение к подпрограмме рисования нот посредством оператора `GOSUB 300`. `CASE 3` и `CASE 4` удалось объединить, вынеся различия — разные параметры счетчика вложенного цикла — в строку 60 с условным оператором `IF...THEN`. Причем `CASE 3` (счетчик цикла `i` изменяется от 1 до 8 с шагом 1) обеспечивает звучание гаммы в прямом порядке, а `CASE 4` (счетчик цикла `i` изменяется от 8 до 1 с шагом –1) — в обратном.

Линии нотного стана строятся посредством оператора `LINE` в строке 210 подпрограммы. Оператор `LINE (x1, y1)-(x2, y2)` рисует отрезок прямой, выходящий из точки с координатами `x1`, `y1` и заканчивающийся в точке с координатами `x2`, `y2`. Если `y1 = y2`, то это будет, очевидно, отрезок горизонтальной линии *(подробнее об операторе `LINE` см. в разд. 8.1)*. В нашем случае `y1 = y2 = 220 – i * 20`, т. е. линии рисуются с промежутком 20 по вертикали. Поскольку линий в нотном стане 5 плюс дополнительная линия для ноты До — итого 6, а число повторений в цикле, задаваемое в строке 60, равно 8 — по числу нот, то графиче-

ский оператор LINE в строке 210 поставлен в условный оператор IF...THEN. Условный оператор IF...THEN в строке 200 определяет длину линий, рисуемых оператором LINE в строке 210 (короткая для ноты До и пять длинных остальных).

Сами ноты рисуются в соответствующей подпрограмме графическим оператором CIRCLE в строке 300, а штили к ним — оператором LINE в строке 320. Поскольку штили — это вертикальные линии, то STEP по x во второй скобке равен 0. В строке 310 условный оператор IF...THEN с помощью множителя b задает перенаправление рисования штилей (STEP по y во второй скобке оператора LINE), а также меняет начало рисования (по x в первой скобке).

Интересно сравнить рассмотренную программу 1.4 с аналогичной программой, написанной на языке Pascal:

```
Program gamma;
uses Graph, Crt;
var
     grdr, grreg, x1, y1, i: integer;
Procedure nota (x, y, col: integer);
begin
  SetColor (col);
  Ellipse (x, y, 0, 360, 25, 20);
  Ellipse (x, y, 0, 360, 24, 19);
  If x <= 400 then
    begin
        Line (x+25, y, x+25, y-110);
        Line (x+26, y, x+26, y-110);
    end
        else
    begin
        Line (x-25, y, x-25, y+110);
        Line (x-24, y, x-24, y+110);
    end
end;
Procedure zvuk (n: integer);
begin
  Sound (n);
  nota (x1, y1, green);
  Delay (20000);
  NoSound;
  Delay (10000);
  nota (x1, y1, black);
  x1 := x1+60; y1 := y1-20
end;
Procedure obrzvuk (n: integer);
begin
  Sound (n);
  nota (x1, y1, green);
  Delay (20000);
  NoSound;
  Delay (10000);
  nota (x1, y1, black);
  x1 := x1-60; y1 := y1+20
end;
```

```
begin
  grdr := detect;
  InitGraph (grdr, grreg, 'd:\tp7\bgi');
  SetFillStyle (solidfill, lightgray);
  FloodFill (100, 100, lightgray);
  SetColor (black);
  Line (10, 100, 630, 100);
  Line (10, 140, 630, 140);
  Line (10, 180, 630, 180);
  Line (10, 220, 630, 220);
  Line (10, 260, 630, 260);
  Line (65, 300, 135, 300);
  x1 := 100; y1 := 300;
  For i := 1 to 8 do
      begin
          nota (x1, y1, black);
          x1 := x1+60; y1 := y1-20
      end;
  SetColor (blue);
  SetTextStyle (defaultfont, horizdir, 2);
  OutTextXY (100, 400, 'Для воспроизведения гаммы');
  OutTextXY (100, 430, 'нажмите клавишу "Enter"');
  Readln;
  Bar (0, 400, 639, 480);
  x1:= 100; y1 := 300;
  zvuk (262); zvuk (294);
  zvuk (330); zvuk (349);
  zvuk (392); zvuk (440);
  zvuk (494); zvuk (523);
  x1:= 520; y1 := 160;
  obrzvuk (523); obrzvuk (494);
  obrzvuk (440); obrzvuk (392);
  obrzvuk (349); obrzvuk (330);
  obrzvuk (294); obrzvuk (262);
  Readln;
  Closegraph
end.
```

Первое, что бросается в глаза, — это количество строк в программах. Если их посчитать, то программа на Pascal почти вдвое длиннее, чем на BASIC. Это происходит за счет применения в Pascal операторных скобок `begin...end`, обязательного описания переменных и других конструкций языка. Отметим, что в BASIC описание переменных, за исключением массивов, следует применять лишь в тех случаях, когда, например, не хватает быстродействия или памяти. Массивы же рекомендуется описывать посредством операторов `DIM` или `REDIM`, хотя по умолчанию размер массива в BASIC принимается равным 10. Так что строка 40 в программе 1.4 не является обязательной.

Теперь, "звуковые" команды в BASIC задают как частоту, так и длительность звучания. В Pascal же для определения времени звучания используется стандартная процедура `Delay`.

Для рисования нот в программе на языке BASIC разработана подпрограмма в строках 300—330. В программе на Pascal организуется специальная процедура. Далее, в программе на

Pascal для воспроизведения гаммы в прямом порядке создана процедура `zvuk` и в обратном порядке — процедура `obrzvuk`. В программе 1.4 на BASIC это удалось объединить в одном цикле путем переопределения значений счетчика цикла в строках 90—120. Алгоритм же решения поставленной задачи практически одинаков как на BASIC, так и на Pascal.

Следующий пример из математики.

Пример 1.5. Таблица умножения

Разработать программу, выводящую таблицу умножения.

Программа 1.5

```
'1TabUmn.bas        Программа, выводящая таблицу умножения
10 CLS
20 mx = 1
30 PRINT SPC(27); "Таблица умножения"
40 FOR j = 1 TO 9
50 IF j <= 5 THEN my = 2: k = 1 ELSE my = 12: k = 6
60 FOR i = 1 TO 9
70 LOCATE my + i, mx + (j - k) * 15
80 PRINT j; "x"; i; "="; j * i
90 NEXT i, j
100 END
```

Результат работы программы 1.5 приведен на рис. 1.8.

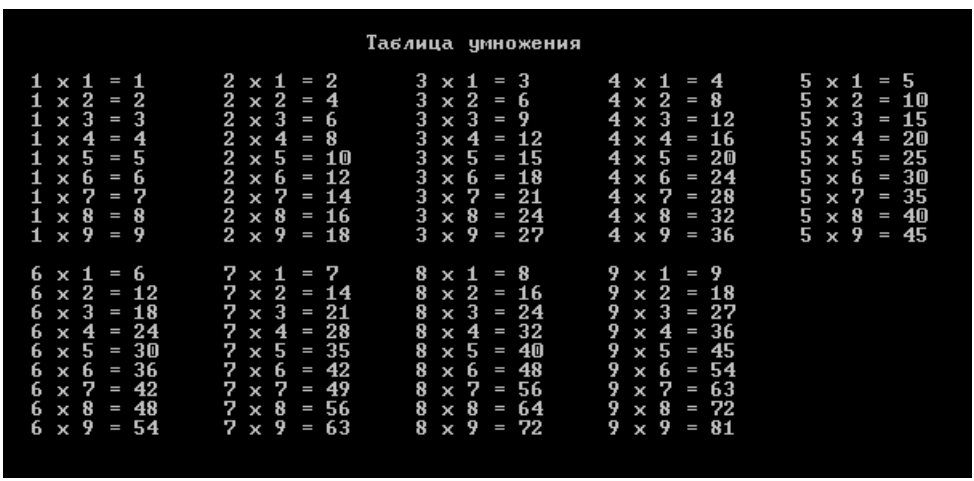


Рис. 1.8. Вывод программы "Таблица умножения"

Как нетрудно заметить, программа 1.5 простая, и при ее разработке вполне можно обойтись и без блок-схемы. Впрочем, если читатель с этим не согласен, то может разработать ее самостоятельно. Авторы полагают, что это вполне по силам, особенно если перед этим предварительно прочитать *разд. 1.2*, посвященный разработке блок-схем.

Поскольку по условию задания требуется вывести 9 столбцов по 9 строк в каждом, то для решения задачи организуется двойной (вложенный) цикл `FOR...NEXT` в строках 40—90.

Причем внешний цикл обеспечивает вывод нужного числа столбцов, а вложенный FOR...NEXT в строках 60—90 реализует вывод строк в столбцах таблицы умножения.

Вывод строк обеспечивает пара операторов LOCATE и PRINT в строках 70 и 80 соответственно. Причем LOCATE задает позицию вывода столбца, а PRINT определяет вид выводимой строки. В строке 70 стоит оператор следующего вида:

```
LOCATE my + i, mx + (j — k) * 15
```

Здесь my + i задает позицию по y, поскольку в цикле на каждом шаге i увеличивается на 1, то каждая последующая строка столбца таблицы умножения будет выводиться с новой строки.

А mx + (j — k) * 15 задает позицию по x, причем (j — k) * 15 определяет расстояние между столбцами таблицы умножения.

Так как столбцы выводятся в два ряда (в верхнем ряду 5 столбцов и в нижнем — 4), то с помощью условного оператора IF...THEN в строке 50 задается смещение по вертикали позиций начала вывода столбцов нижнего ряда. Причем для того, чтобы шестой столбец таблицы умножения и первый в нижнем ряду располагался под первым столбцом таблицы умножения и первым в верхнем ряду, в строке 50 задается изменение значения k с 1 на 6.

Следующий пример также из математики, но с элементами псевдографики. Здесь под псевдографикой понимается рисование, например, таблицы без включения графического режима.

Пример 1.6. Таблица значений функции

Разработать программу, выводящую таблицу значений функции $\sin x/x$ при x в диапазоне от 0.1 до 0.9 с шагом 0.1.

Блок-схема алгоритма решения задачи представлена на рис. 1.9. На ее основе разработана программа 1.6, результат работы которой приведен на рис. 1.10.

Программа 1.6

```
'1sinx.bas             Таблица значений функции sinx/x
10 CLS
20 DIM mx(22), my(22)
30 LOCATE 4, 3
40 PRINT "  x"                      'название верхней строки таблицы
50 LOCATE 6, 3
60 PRINT "sinx/x "                  'название нижней строки таблицы
'==1===== Блок вычисления функции ==================================
70 FOR x = .1 TO 1 STEP .1
   '---1.1----- вывод x -------------------------------------------
80 LOCATE 4, 5 + 6 * x / .1
90 PRINT USING "#.#"; x
   '---1.2----- вычисление и вывод sinx/x -------------------------
100 y = SIN(x) / x
110 LOCATE 6, 4 + 6 * x / .1
120 PRINT USING "#.###"; y
130 PRINT " ";
140 NEXT x
```

```
'==2===== Блок вывода горизонтальных линий ========================
150 LOCATE 3, 3: PRINT STRING$(60, 196)
160 LOCATE 5, 3: PRINT STRING$(60, 196)
170 LOCATE 7, 3: PRINT STRING$(60, 196)
'==3===== Блок вывода вертикальных линий ==========================
180 FOR j = 1 TO 11
190 DATA 2,9,15,21,27,33,39,45,51,57,63
200 READ mx
210 FOR i = 4 TO 6
220 LOCATE i, mx: PRINT CHR$(179)
230 NEXT i, j
'==4===== Блок вывода соединительных элементов ====================
240 FOR j = 1 TO 22
250 READ my(j), mx(j), kod
260 LOCATE my(j), mx(j): PRINT CHR$(kod)
270 NEXT j
280 DATA 3,2,218, 3,9,194, 3,15,194, 3,21,194, 3,27,194, 3,33,194
290 DATA 3,39,194, 3,45,194, 3,51,194, 3,57,194, 3,63,191
300 DATA 7,2,192, 7,9,193, 7,15,193, 7,21,193, 7,27,193, 7,33,193
310 DATA 7,39,193, 7,45,193, 7,51,193, 7,57,193, 7,63,217
'================================================================
320 END
```

Как видно из блок-схемы, программа должна состоять из четырех блоков:

- блок вычисления значений функции `y` и вывода значений `x` и `y` (блоки 1—6 блок-схемы, строки 70—140 программы);
- блок рисования трех горизонтальных линий таблицы (блоки 7—10 и строки 150—170 программы);
- блок рисования одиннадцати вертикальных линий таблицы (блоки 11—14 и строки 180—230);
- блок рисования 22-х соединительных элементов таблицы (блоки 15—18 блок-схемы и строки 240—310 программы 1.6).

Чтобы вывести значения `x` и функции таблицей, в строках 90 и 120 используются операторы форматированного вывода `PRINT USING`, позволяющие задать количество знаков после запятой (здесь точки). Сколько решеток поставлено в скобках после точки, столько знаков и будет выведено. Подробнее оператор `PRINT USING` будет рассмотрен в *разд. 4.3*.

Кроме операторов `PRINT USING` для нужного вывода необходимо использовать операторы `LOCATE`, определяющие позицию вывода. Поэтому они использованы в строках 80 и 110. Например, в строке 80 программы оператор `LOCATE` выглядит следующим образом:

```
LOCATE 4, 5 + 6 * x / .1
```

где `4` — позиция вывода по вертикали (по `y`), а `5 + 6 * x / .1` — позиция вывода по горизонтали (по `x`). Причем `6 * x / .1` задает промежуток между соседними позициями вывода по горизонтали.

Следующие три блока программы 1.6 являются реализацией псевдографики с целью построения таблицы. Для рисования трех горизонтальных линий использована функция `STRING$(n, kod)`, заполняющая строку заданным символом, где `n` — количество символов в строке, а `kod` — код символа, которым следует заполнить строку. В строках 150—170 после

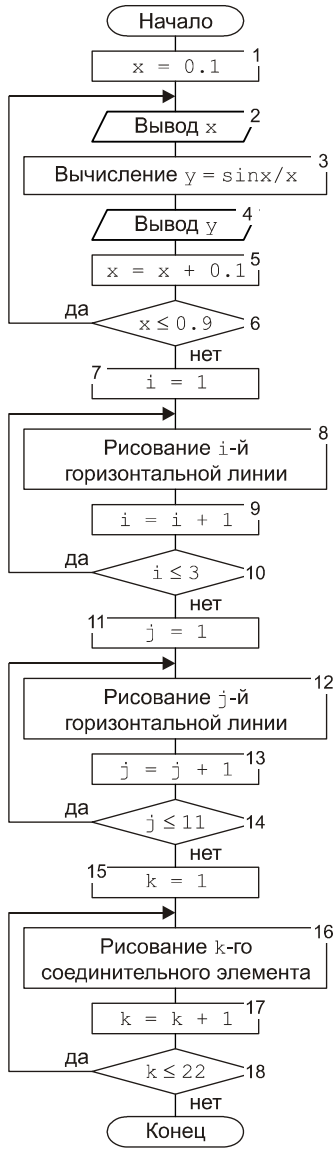


Рис. 1.9. Блок-схема математической программы

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
sinx/x	0.998	0.993	0.985	0.974	0.959	0.941	0.920	0.897	0.870

Рис. 1.10. Таблица значений функции $y = \sin x / x$

оператора PRINT поставлена функция STRING$(60, 196), выводящая строку из 60 горизонтальных одинарных элементов псевдографики, код которых 196.

Принцип построения вертикальных линий несколько другой. Для построения вертикальной линии требуется вывести по вертикали (по y) нужное количество вертикальных одинарных элементов псевдографики, код которых 179. С этой целью организуется цикл FOR...NEXT в строках 210—230, в теле которого находятся операторы LOCATE и PRINT. Именно оператор LOCATE обеспечивает вывод вертикальных элементов псевдографики друг под другом. После оператора PRINT стоит функция CHR$, выводящая элемент псевдографики с заданным кодом.

Этот цикл, являющийся вложенным, выводит одну вертикальную линию. Чтобы нарисовать все 11 требуемых вертикальных линий, организован внешний цикл FOR...NEXT в строках 180—230. Пара операторов READ и DATA в строках 190—200 задает позиции по горизонтали (по x) этих линий.

Последний, четвертый, блок программы 1.6 служит для вывода соединительных элементов псевдографики, под которыми понимаются уголки и пересечения. Цикл FOR...NEXT в строках 240—270 обеспечивает рисование этих элементов. В операторах DATA в строках 280—310 данные организованы по тройкам — значения позиций по y и x соответственно, для позиционирующего оператора LOCATE в строке 260 и код элемента для функции CHR$ в той же строке. Именно так, по тройкам, снимаются данные оператором READ в строке 250. В операторах DATA содержатся коды следующих одинарных элементов псевдографики:

- верхний левый уголок — код 218;
- верхний средний уголок — код 194;
- верхний правый уголок — код 191;
- нижний левый уголок — код 192;
- нижний средний уголок — код 193;
- нижний правый уголок — код 217.

Вообще-то соединительных элементов должно быть 33 — еще 11 для средней горизонтали, без которых она является прерывистой. Авторам показалось, что таблица и так выглядит довольно прилично. Читателям-эстетам, не согласным с авторами, предлагается замкнуть среднюю горизонталь, доработав программу. Для них сообщаем, что код среднего левого уголка — 195, код пересечения — 197 и код среднего правого уголка — 180.

Следующий пример на разработку программы, реализующей динамическую графику (анимацию).

Пример 1.7. Погружение в Бейсик

Разработать программу, рисующую динамическую картинку-шутку "Погружение в среду Бейсика".

Блок-схема алгоритма решения задачи приведена на рис. 1.11.

Результат работы программы 1.7 показан на рис. 1.12.

Программа 1.7

```
'1Basic.bas    Погружение в Бейсик
10 CLS                      'очистка экрана
20 SCREEN 9                 'установка экранного режима
```

```
30 COLOR 8, 7                'установка экранных цветов (7 — цвет фона)
40 DIM yk(8)                 'готовит место в памяти для конечных точек
'==1===== Блок начальных установок ================================
50 LINE (0, 0)-(650, 250), 5, BF
   '--1.1----- Груз заданий ---------------------------------------
60 LINE (290, 0)-(310, 30), 0, BF
70 LINE STEP(-60, 0)-STEP(100, 40), 0, BF
   '--1.2----- Конечные точки груза -------------------------------
80 LOCATE 21, 33: PRINT "Б Е Й С И К"
90 yk(1) = 100: yk(2) = yk(1) + 20: yk(3) = yk(1) + 40: yk(4) = yk(1) + 60
'==2===== Блок движения ============================================
100 y1 = 160: GOSUB 500                    'рисует студента
110 SLEEP 2
120 FOR k = 1 TO 4
130 y = 1: dy = 1
140 DO
150 LINE (290, y)-STEP(20, 30), 0, BF      'рисует груз заданий
160 LINE STEP(-60, 0)-STEP(100, 40), 0, BF
170 FOR iv = 1 TO 100: NEXT iv             'выдержка времени
   '--2.1----- Стирает груз заданий -------------------------------
180 IF dy = 1 THEN PSET (250, y + 29), 5: DRAW "c5 r39 br22 r39"
190 IF dy = -1 THEN LINE (250, y + 71)-(350, y + 71), 5
   '--2.2----- Анализ -----------------------------------------------
200 IF y = yk(k) - 21 AND dy = 1 THEN GOSUB 400      'стирает студента
210 IF y = yk(k) - 21 AND dy = 1 THEN y1 = yk(k) + 80: GOSUB 500       'рисует студента
220 IF y = yk(k) THEN  ELSE 240    'перемена направления движения груза
230 dy = -dy: LOCATE 11 + k, 34: PRINT "Задание"; k: SLEEP 2
240 IF y = 30 AND dy = -1 THEN EXIT DO              'выход из цикла
250 y = y + dy
260 LOOP
270 NEXT k
280 GOTO 600
'==3===== Подпрограмма стирания студента ===========================
400 LINE (0, 249)-(650, 249), 5: PAINT (300, 248), 5, 5
410 RETURN
'==4===== Подпрограмма рисования студента ==========================
500 CIRCLE (300, y1), 12, 14: PAINT STEP(0, 0), 14, 14        'голова
510 IF y1 <= 220 THEN  ELSE 540
520 DRAW "c1 bm-3,+10 r6 m+3,+1 f4 m+2,+3 d12 l24 u12 m+2,-3"
530 DRAW "e4 m+3,-1 bd9 p1,1"
540 IF y1 <= 200 THEN DRAW "bm-9,+11 nr24 d20 r24 nu20 bm-2,-1 p1,1"
550 IF y1 <= 180 THEN DRAW "m+0,+1 nl20 m-1,+20 l18 nm-1,-20 be1 p1,1"
560 IF y1 <= 160 THEN DRAW "bg1 nr18 m+1,+20 r16 nm+1,-20 bh1 p1,1"
570 RETURN
'===================================================================
600 END
```

В основе создания динамических изображений лежит следующий принцип — выводится рисунок, который после определенной временной задержки стирается, чтобы тотчас появиться в другом заданном месте. Этот принцип получил отражение в блоках 5—7 блок-схемы на рис. 1.11 и в строках 150—190 программы 1.7. Поскольку груз знаний является комбинацией простых геометрических фигур (два прямоугольника), мало изменяющихся в процессе движения, то для создания движения целесообразно использовать циклы.

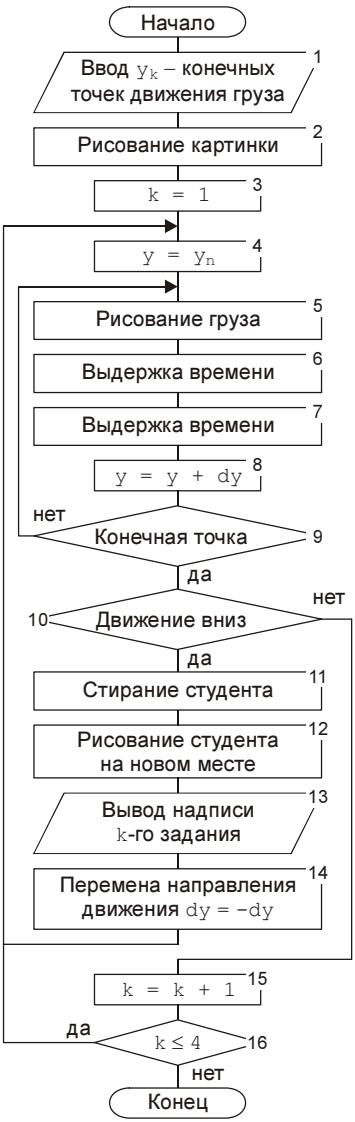


Рис. 1.11. Блок-схема программы "Погружение в Бейсик"

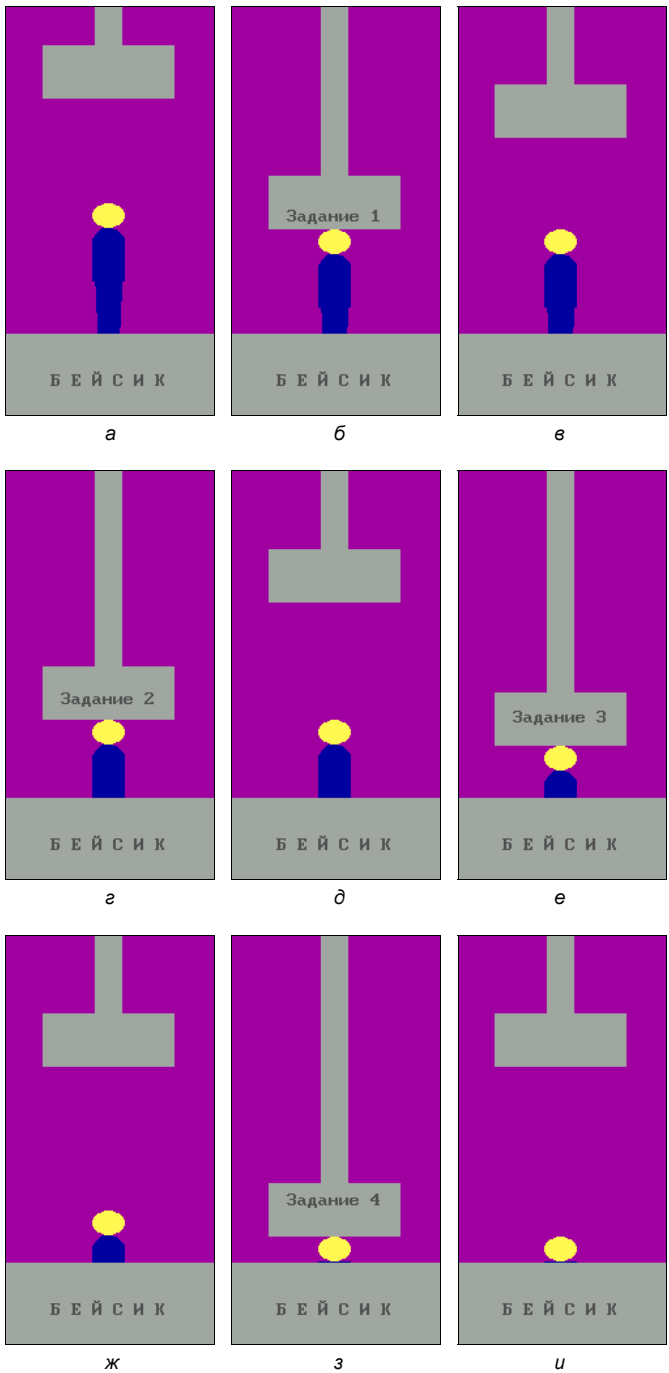


Рис. 1.12. Погружение в среду Бейсика

Цикл, задающий движение груза заданий, отображен в блоках 4—14 блок-схемы. В рассматриваемой программе 1.7 с этой целью организован цикл с оператором цикла DO...LOOP в строках 140—260.

Итак, перед дальнейшим рассмотрением программы 1.7 сформулируем, что же она должна делать (рис. 1.12).

На 1-м шаге выполняется проход груза заданий вниз до стоящего студента, который при проходе груза стирается и тут же выводится, погруженным в среду Бейсика на четверть туловища. На грузе заданий выводится надпись "Задание 1", после чего он возвращается вверх в исходное положение.

На 2-м шаге — проход груза вниз, погружение студента на половину туловища в среду, вывод на грузе надписи "Задание 2" и его возврат в исходное состояние.

На 3-м шаге — проход груза вниз, погружение студента на три четверти туловища, вывод на грузе надписи "Задание 3" и его возврат в исходное состояние.

На 4-м завершающем шаге — проход груза вниз, погружение студента по горлышко, вывод на грузе надписи "Задание 4" и его возврат в исходное состояние. После этого программа завершает работу.

В соответствии с вышеизложенным, следует создать цикл с числом повторений — 4. В блок-схеме это блоки 3, 15—16. В программе 1.7 заданное число повторений обеспечивает оператор цикла FOR...NEXT в строках 120—270. Груз заданий рисуется с помощью двух графических операторов LINE с параметром BF в строках 150—160. Графический оператор LINE с параметром BF рисует прямоугольник и закрашивает его. Посредством пустого цикла FOR...NEXT в строке 170 задается выдержка времени. Очевидно, чем больше число повторений в пустом цикле, тем больше выдержка времени.

Для создания движения стирать графический объект можно либо полностью, либо частично. Во втором случае обычно стираются задние фронты изображения объекта. Именно такой подход и реализован в программе 1.7. При движении груза заданий вниз (dy = 1) стираются два его плеча командами r39 графического оператора DRAW в строке 180, а при движении вверх (dy = –1) задний фронт стирается оператором LINE в строке 190. Стирание задних фронтов позволяет получать качественное изображение движущегося объекта без возможного мигания, которое случается, если объект стирается полностью. Однако такой подход достаточно просто реализуется лишь при прямолинейном движении несложного по конфигурации объекта, такого как груз заданий. Графические операторы LINE, DRAW и CIRCLE будут подробно рассмотрены в *разд. 8.1*.

Блок 1 блок-схемы на рис. 1.11 реализуется в программе посредством операторов присваивания в строке 90, а блок 2 (рисование картинки) — с помощью графических операторов LINE в строках 50—70 и пары операторов LOCATE и PRINT в строке 80. Отметим, что оператор LINE в строке 50 изменяет цвет фона в верхней половине экрана.

"Погружение" студента в среду Бейсика обеспечивают условные операторы IF...THEN в строках 200—210, выполняющие по условию сначала обращение к подпрограмме стирания студента (GOSUB 400), а затем к подпрограмме рисования студента (GOSUB 500). При достижении грузом заданной конечной точки при движении вниз выполняется условие в условном операторе IF...THEN в строке 220, происходит изменение направления движения (dy = –dy) и выводится надпись "Задание к", где к = 1, 2, 3, 4, соответственно шагам внешнего цикла. При достижении грузом заданной конечной точки при движении вверх выполняется условие в условном операторе IF...THEN в строке 240, тем самым выходом из вложенного цикла DO...LOOP завершается очередной шаг внешнего цикла FOR...NEXT. Напомним, что оператор безусловного перехода GOTO 600 в строке 280 необходим для того, чтобы исключить обра-

щение к подпрограмме без GOSUB. В противном случае появится сообщение об ошибке "RETURN без GOSUB".

В подпрограмме стирается изображение студента посредством графического оператора PAINT (300, 248), 5, 5 в строке 400. Оператор производит закрашивание цветом 5 (первая 5-ка) замкнутого контура, ограниченного цветом 5 (вторая 5-ка), причем координаты точки начала закраски (300, 248) должны принадлежать области, ограниченной этим контуром. Как видно из рис. 1.11, *а*, изображение студента со всех сторон окружено областью цветом 5 (сиреневый), за исключением нижней части, примыкающей к среде Бейсика. Поэтому такое необходимое ограничение осуществляется линией, рисуемой оператором LINE в строке 400. Завершается подпрограмма соответствующим служебным словом RETURN.

Особенностью подпрограммы рисования студента является то, что по мере "погружения" студента в среду он рисуется не полностью. Выводится лишь "непогруженная" часть фигуры. Голова студента рисуется операторами CIRCLE и PAINT в строке 500, туловище и ноги — операторами DRAW.

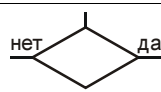
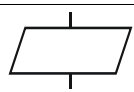
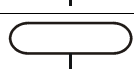
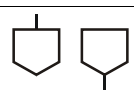
Итак, уважаемый читатель, вы проработали семь примеров разных видов, приобретя, тем самым, начальные сведения, необходимые для программирования на языке Basic. Возможно, их вам будет достаточно для решения своих задач и для создания собственных программ. Если же потребуется что-нибудь еще, то к вашим услугам другие разделы этой книги.

1.2. Разработка блок-схемы

1.2.1. Разработка машинно-ориентированного алгоритма

Под машинно-ориентированным понимается алгоритм, удобный для решения данной задачи на ЭВМ. Это очень важный этап, т. к. алгоритм определяет логическую структуру программы. Алгоритм может быть описан словесно или графом (называемым блок-схемой), что строже и нагляднее. Словесное описание применяется обычно для разрешения затруднений при построении графа. Граф состоит из вершин (блоков), объединенных ребрами. Типы блоков представлены в табл. 1.3.

Таблица 1.3. *Основные блоки схем алгоритмов*

Наименование	Обозначение	Функция
Процесс		Выполнение действия или действий, изменяющих значение, форму представления или расположение данных
Решение		Выбор направления выполнения алгоритма в зависимости от некоторых переменных условий
Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или для отображения результатов обработки (вывод)
Пуск-останов		Начало или конец процесса обработки данных
Межстраничный соединитель		Указание связи между разъединенными частями схемы алгоритма, расположенными на разных листах

К вершинам блока типа 1 и 3 подходят два ребра (одно входящее и одно выходящее). К вершинам типа 4 и 5 — лишь одно ребро (либо входящее, либо выходящее). В табл. 1.3 для типа 4 представлен вариант блока "Начало". К вершинам типа 2 подходят три ребра (одно входящее и два выходящих — для "да" и "нет"), причем одно из выходящих ребер может начинаться из нижнего угла ромба. Таким образом, вершине типа 2, в зависимости от расположения выходящих ребер и сопоставленных им "да" и "нет", соответствуют 6 вариантов. Вершины типа 1—3 в блок-схеме обычно нумеруются. Подразумевается, что движение по графу выполняется сверху вниз. При соблюдении этого правила стрелки не используются, а иное направление указывается ребром со стрелкой.

В соответствии с основной теоремой структурного программирования, доказанной Э. Дейкстрой, алгоритм любой сложности можно реализовать, используя только три конструкции: следование (оператор за оператором), повторение (цикл), выбор (альтернатива). Первой конструкции соответствует линейная программа (пример алгоритма, который приведен на рис. 1.13, *а*), второй — циклическая (рис. 1.13, *б*), третьей — с разветвлениями (рис. 1.13, *в*).

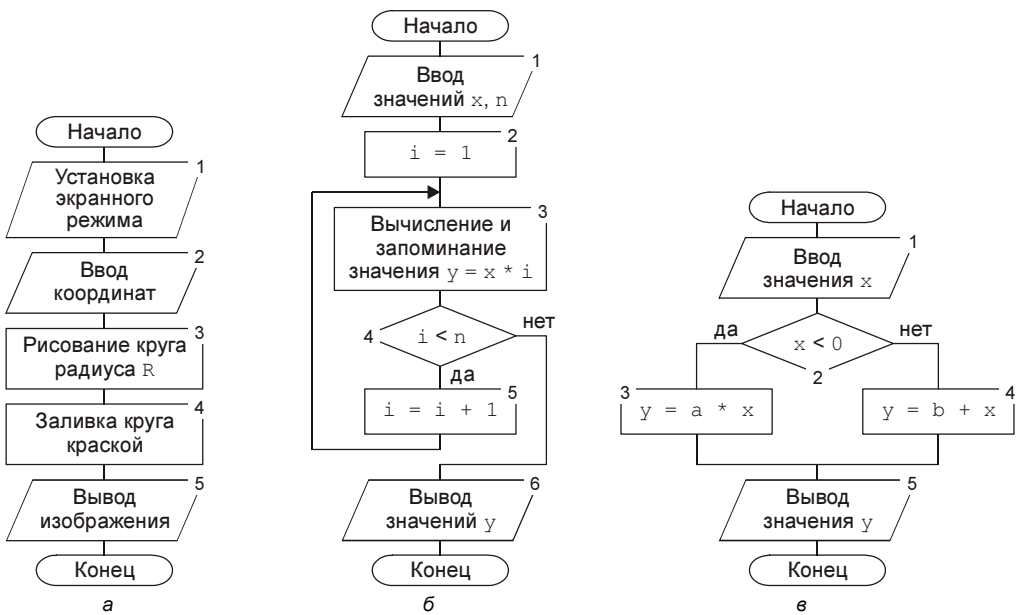


Рис. 1.13. Варианты реализации алгоритма любой сложности

В укрупненном виде алгоритм состоит из:

1. Ввода входных данных.
2. Алгоритма программы, представляющего собой необходимую комбинацию из трех конструкций.
3. Вывода результата.

Ввод входных данных и вывод результата тоже могут иметь сложную структуру.

Построим блок-схему для примера экономической задачи из *разд. 1.1*.

Пример 1.8. Алгоритмы экономической задачи

Разработать алгоритмы расчета времени накопления заданной суммы средств *Kk* от начальной суммы вклада *Kn*, если процентная ставка равна *p*, при условии начисления простых и сложных ежегодных процентов (см. пример 1.1). Имеются те же формулы для вычисления суммы при заданных видах процентов, что и в примере 1.1:

$S = Kn \cdot (1 + t \cdot p/100)$ — для простых процентов,

$S = Kn \cdot (1 + p/100)^t$ — для сложных процентов.

Сначала составим словесный алгоритм. Словесный алгоритм лучше составлять на бумаге. В простых случаях можно и в уме.

1. Задать номер варианта:
   - Простые проценты — вариант 1;
   - Сложные проценты — вариант 2;
   - Сравнение вариантов 1 и 2 — вариант 3.
2. Ввести значения исходных данных:
   - `Kn` — начального вклада;
   - `Kk` — накопляемой суммы;
   - `p` — годовой процентной ставки.
3. Обеспечить выбор одного из трех вариантов.
4. Вычислить простые проценты.
5. Вычислить сложные проценты.
6. Сравнить варианты 1 и 2.
7. Вывести результат.

Обозначим за `t1` — время для накопления простых процентов, а текущую накапливаемую сумму через `S1`. Теперь строим цикл для вычисления простых процентов (рис. 1.14, *а*), для чего блок с формулой помещается в обрамление цикла: задается начальное значение для счетчика цикла `t1 = 1`, после блока с формулой следует блок приращения счетчика цикла `t1 = t1 + 1`, и завершается этот фрагмент проверкой условия выхода из цикла `S1 ≥ Kk`.

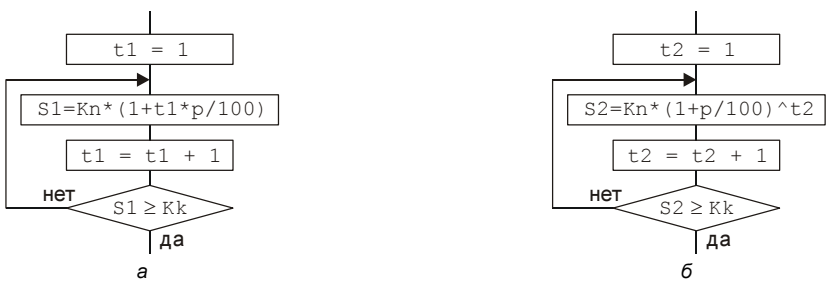


Рис. 1.14. Цикл для вычисления простых процентов (*а*) и цикл для вычисления сложных процентов (*б*)

Аналогично строится цикл для вычисления сложных процентов, в котором изменяется лишь блок с формулой (рис. 1.14, *б*).

Теперь, добавив блок сравнения вариантов `t = t1 – t2`, следует объединить эти фрагменты блоками выбора варианта (рис. 1.15).

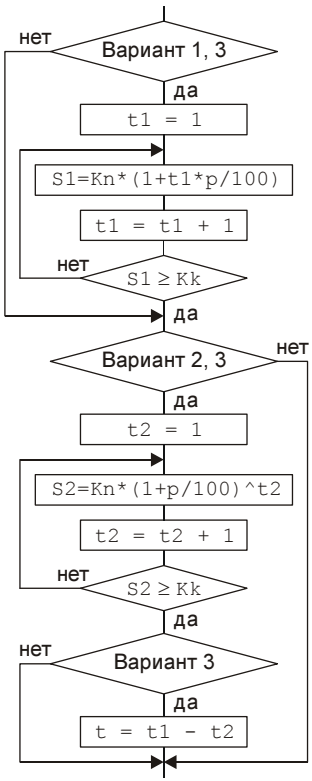


Рис. 1.15. Объединение вариантов блоками выбора

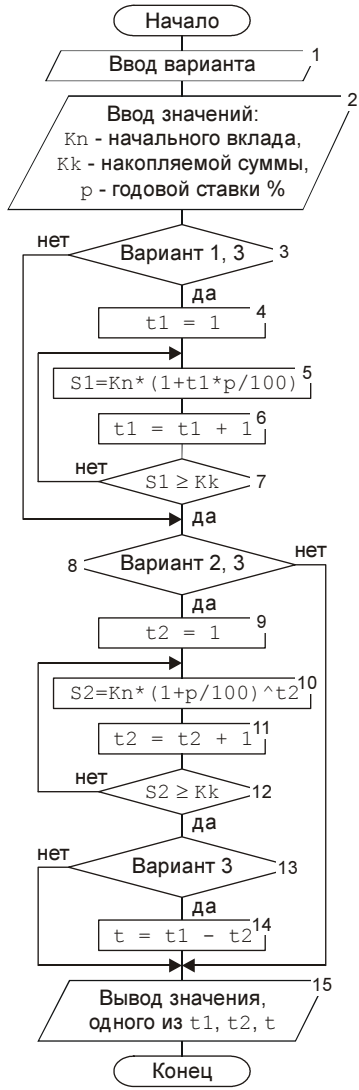


Рис. 1.16. Блок-схема экономической задачи

Теперь, добавив в начале два блока ввода и блок начала, а в конце — блок вывода и блок конца, и пронумеровав блоки, получим окончательный вид разрабатываемой блок-схемы (рис. 1.16).

Хорошо составленный алгоритм позволяет проверять правильность работы программы без компьютера. Для этого необходимо добросовестно пройтись по блокам алгоритма. Рассмотрим пример прохода по блокам алгоритма.

Блок 1: выбираем вариант 1.

Блок 2: задаем значения начального вклада Kn = 1000 руб., накопляемой суммы Kk = 2000 руб. и ставки годовых процентов p = 20%.

Блок 3: определяется путь по "да". Начинается цикл по вычислению времени накопления с простыми процентами.

Блок 4: t1 = 1.

Шаг 1. Блок 5: вычисляется S1 = 1200. Блок 6: вычисляется t1 = t1 + 1 = 2. Блок 7: проверка (S1 = 1200) < (Kk = 2000), по "нет".

Шаг 2. Блок 5: вычисляется S1 = 1400. Блок 6: вычисляется t1 = t1 + 1 = 3. Блок 7: проверка (S1 = 1400) < (Kk = 2000), по "нет".

Шаг 3. Блок 5: вычисляется S1 = 1600. Блок 6: вычисляется t1 = t1 + 1 = 4. Блок 7: проверка (S1 = 1600) < (Kk = 2000), по "нет".

Шаг 4. Блок 5: вычисляется S1 = 1800. Блок 6: вычисляется t1 = t1 + 1 = 5. Блок 7: проверка (S1 = 1800) < (Kk = 2000), по "нет".

Шаг 5. Блок 5: вычисляется S1 = 2000. Блок 6: вычисляется t1 = t1 + 1 = 6. Блок 7: проверка (S1 = 2000) = (Kk = 2000), по "да".

Завершен цикл по вычислению времени накопления.

Блок 8: по "нет".

Блок 15: вывод времени накопления t1 – 1 = 5 лет.

В результате проверки выявилась некоторая погрешность блок-схемы, поскольку время накопления t1 = 5 лет, а на выходе 5-го шага цикла t1 = 6. Поэтому приходится выводить t1 – 1. А лучше скорректировать блок-схему. Так, фрагмент для вычисления простых процентов будет выглядеть таким образом, как показано на рис. 1.17.

Аналогично следует скорректировать и цикл для вычисления сложных процентов. Тогда модернизированная блок-схема представлена на рис. 1.18.

Теперь пройдемся по блокам.

Блок 1: выбираем вариант 2.

Блок 2: задаем значения начального вклада Kn = 1000 руб., накопляемой суммы Kk = 2000 руб. и ставки годовых процентов p = 20%.

Блок 3: определяется путь по "нет".

Блок 8: определяет путь по "да".

Начинается цикл по вычислению времени накопления со сложными процентами.

Блок 9: t2 = 1.

Шаг 1. Блок 10: вычисляется S2 = 1200. Блок 11: проверка (S2 = 1200) < (Kk = 2000), по "нет". Блок 12: вычисляется t2 = t2 + 1 = 2.

Шаг 2. Блок 10: вычисляется S2 = 1440. Блок 11: проверка (S2 = 1400) < (Kk = 2000), по "нет". Блок 12: вычисляется t2 = t2 + 1 = 3.

Шаг 3. Блок 10: вычисляется S2 = 1728. Блок 11: проверка (S2 = 1728) < (Kk = 2000), по "нет". Блок 12: вычисляется t2 = t2 + 1 = 4.

Шаг 4. Блок 10: вычисляется S2 = 2074. Блок 11: проверка (S2 = 2074) > (Kk = 2000), по "да".

Завершен цикл по вычислению времени накопления.

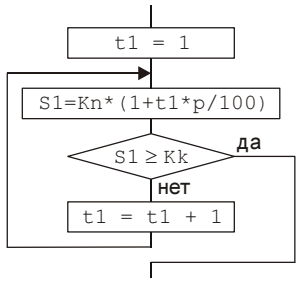


Рис. 1.17. Модифицированный цикл для вычисления простых процентов

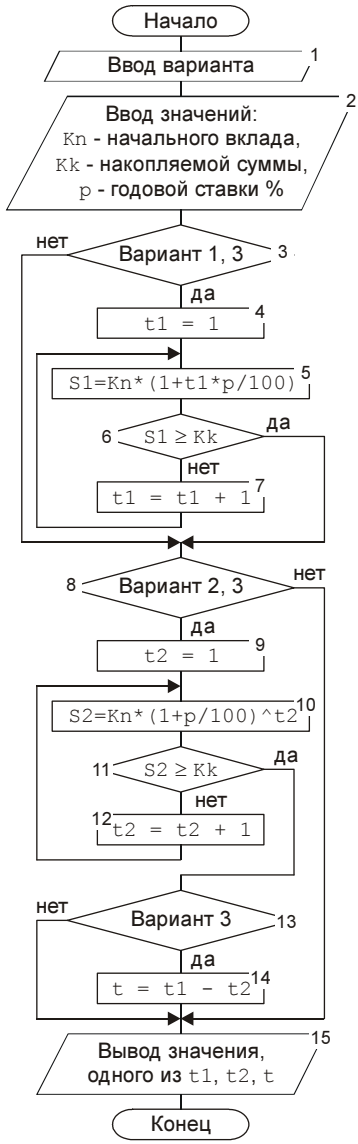


Рис. 1.18. Модифицированная блок-схема экономической задачи

Блок 13: определяет путь по "нет".

Блок 15: вывод времени накопления `t2` = 4 года.

Как видно, погрешность блок-схемы на рис. 1.16 устранена. Для окончания проверки блок-схемы на рис. 1.18 осталось пройти по варианту 3.

Блок 1: выбираем вариант 3.

Блок 2: задаем значения начального вклада `Kn` = 1000 руб., накопляемой суммы `Kk` = 2000 руб. и ставки годовых процентов `p` = 20%.

Блок 3: определяется путь по "да".

Начинается цикл по вычислению времени накопления с простыми процентами.

Блок 4: `t1` = 1.

. . .

Завершен цикл по вычислению времени накопления.

Блок 8: определяет путь по "да".

Начинается цикл по вычислению времени накопления со сложными процентами.

Блок 9: `t2` = 1.

. . .

Завершен цикл по вычислению времени накопления.

Блок 13: определяет путь по "да".

Блок 14: вычисляется `t` = `t1` — `t2` = 1 год.

Блок 15: вывод разницы во времени накопления `t` = 1 год.

Проведенная проверка показала, что блок-схема на рис. 1.18 правильно отражает структуру разрабатываемой программы. Для закрепления полученных навыков построим блок-схему для примера 1.2 из *разд. 1.1*.

Пример 1.9. Диета

Рассмотрим другой возможный вариант решения примера 1.3. Для удобства повторим условие этого примера.

Фирма занимается составлением диеты из пяти продуктов, содержащей по крайней мере 20 единиц белков, 30 единиц углеводов, 10 единиц жиров и 40 единиц витаминов. Как дешевле всего достичь соблюдения диеты при указанных в табл. 1.2 ценах (в рублях) на 1 кг (или 1 л) имеющихся пяти продуктов?

Проще всего реализовать решение задачи путем перебора, тем более что компьютер позволяет это сделать. Итак, словесный алгоритм:

1. Ввод начального значения минимальной стоимости `Cmin`.
2. Цикл по хлебу.
3. Цикл по сое.
4. Цикл по сушеной рыбе.
5. Цикл по фруктам.
6. Цикл по молоку.
7. Условие по белкам.
8. Условие по углеводам.
9. Условие по жирам.
10. Условие по витаминам.
11. Совместность выполнения условий 7—10.
12. Вычисление стоимости пакета продуктов.
13. Проверка стоимости на минимальность.
14. Вывод результата.

Начнем построение блок-схемы с условия по белкам (рис. 1.19, *а*).

Условие по белкам получено из первой строки табл. 1.5. Переключатель понадобится для проверки совместности выполнения всех условий. Аналогично строится фрагмент блок-схемы, реализующий условие по углеводам (рис. 1.19, *б*).

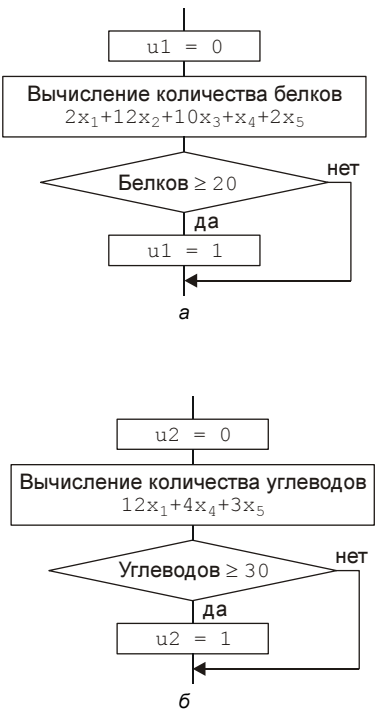


Рис. 1.19. Проверка условия по белкам (*а*) и по углеводам (*б*)

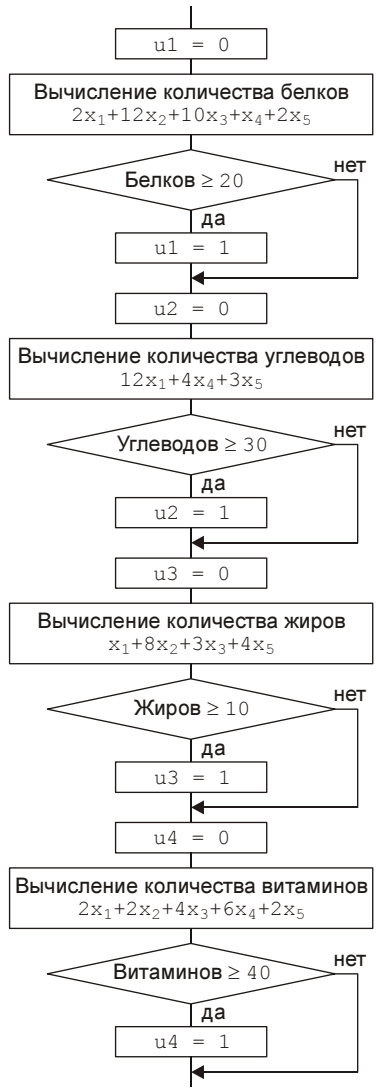


Рис. 1.20. Проверка всех условий

Аналогично строятся фрагменты для проверки условия по жирам и витаминам. Тогда фрагмент по всем условиям будет выглядеть так, как показано на рис. 1.20.

Теперь построим фрагмент проверки одновременности выполнения всех четырех условий (рис. 1.21).

Завершено формирование тела цикла. Заключив построенное тело цикла в циклическое обрамление и добавив в начале блок ввода, а в конце — блок вывода, завершим построение блок-схемы (рис. 1.22).

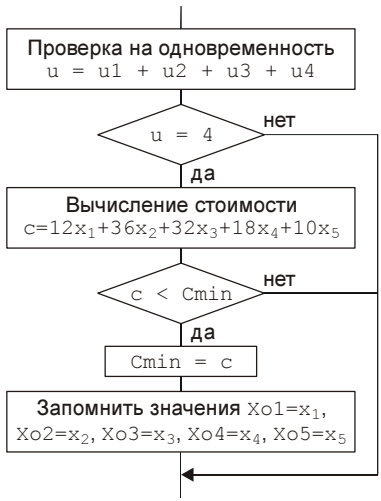
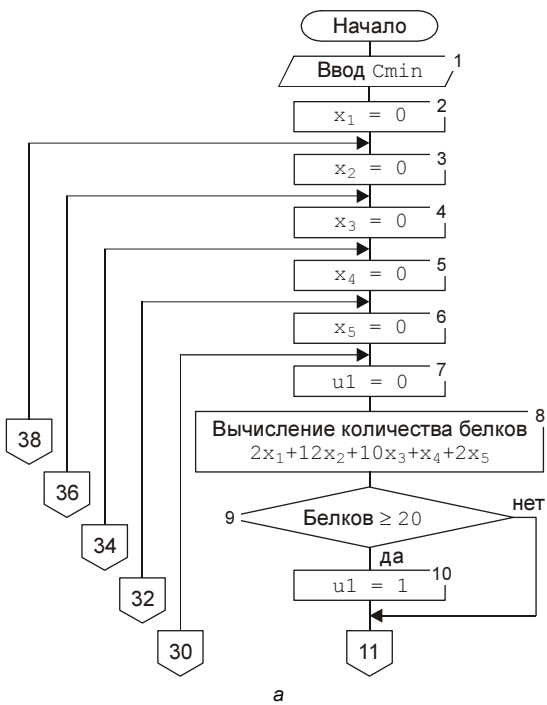


Рис. 1.21. Проверка на одновременность и минимальность



а

Рис. 1.22. Блок-схема задачи оптимизации *(см. продолжение)*

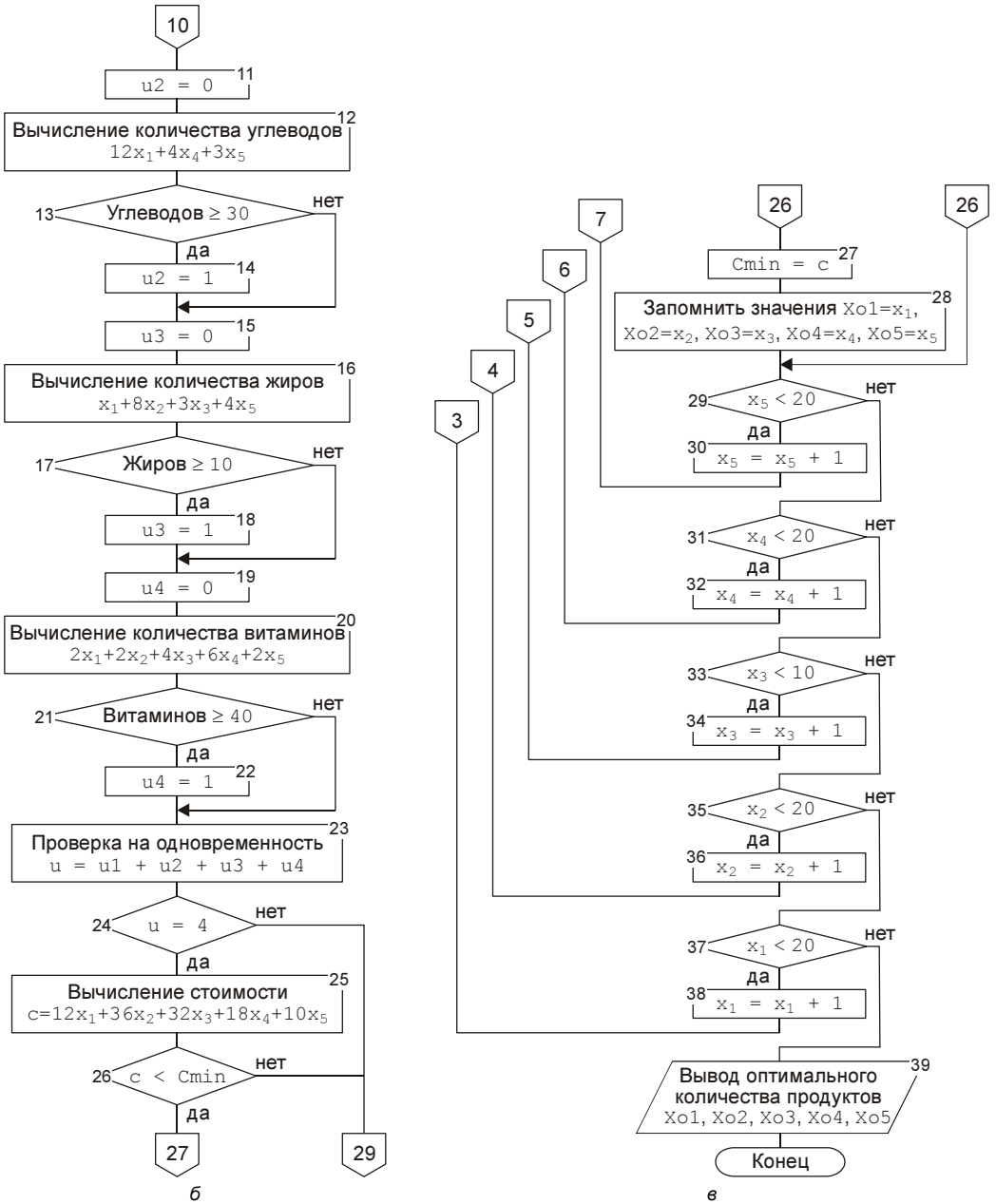


Рис. 1.22. *Окончание*

Пройдемся по логической структуре тела цикла. Пусть мы находимся на шаге с показателями счетчиков цикла $x1 = 4$, $x2 = 12$, $x3 = 10$, $x4 = 15$, $x5 = 15$.

Блок 7: $u1 = 0$.

Блок 8: Вычисление количества белков: $2 * x1 + 12 * x2 + 10 * x3 + x4 + 2 * x5 = 2 * 4 + 12 * 12 + 10 * 10 + 15 + 2 * 15 = 297$.

Блок 9: Белков = 297 > 20, по "да".

Блок 10: u1 = 1.

Блок 11: u2 = 0.

Блок 12: Вычисление количества углеводов: 12 * x1 + 4 * x4 + 3 * x5 = 12 * 4 + 4 * 15 + 3 * 15 = 153.

Блок 13: Углеводов = 153 > 30, по "да".

Блок 14: u2 = 1.

Блок 15: u3 = 0.

Блок 16: Вычисление количества жиров: x1 + 8 * x2 + 3 * x3 + 4 * x5 = 4 + 8 * 12 + 3 * 10 + 4 * 15 = 186.

Блок 17: Жиров = 186 > 10, по "да".

Блок 18: u3 = 1.

Блок 19: u4 = 0.

Блок 20: Вычисление количества витаминов: x1 + 2 * x2 + 4 * x3 + 6 * x4 + 2 * x5 = 4 + 2 * 12 + 4 * 10 + 6 * 15 + 2 * 15 = 188.

Блок 21: Витаминов = 188 > 10, по "да".

Блок 22: u4 = 1.

Блок 23: Проверка на одновременность: u = u1 + u2 + u3 + u4 = 4.

Блок 24: u = 4, по "да".

Блок 25: Вычисление стоимости: c = 12 * x1 + 36 * x2 + 32 * x3 + 18 * x4 + 10 * x5 = 12 * 4 + 36 * 12 + 32 * 10 + 18 * 15 + 10 * 15 = 1220.

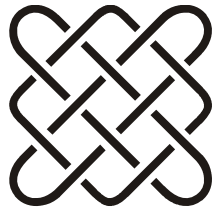
Блок 26: сравниваем стоимость с Cmin. Если c < Cmin, то по "да".

Блок 27: Cmin = c.

Блок 28: Запомнить значения: Xo1 = x1, Xo2 = x2, Xo3 = x3, Xo4 = x4, Xo5 = x5.

Как показала эта небольшая проверка, блок-схема вполне отражает логическую структуру разрабатываемой программы.

Для больших и сложных программ сначала рисуется укрупненный алгоритм, блоки которого потом конкретизируются, вследствие чего алгоритм доводится до машинно-ориентированного вида.



ГЛАВА 2

Трансляция и отладка программы

2.1. Работа со средой QBASIC

Среда программирования QBASIC представляет собой интеллектуальный редактор, осуществляющий синтаксический контроль вводимого текста. Если строка программы, набранная строчными буквами, написана правильно, то после перевода курсора на следующую строку все ключевые слова QBASIC будут написаны заглавными буквами, операторы отделены друг от друга. Это позволяет сразу понять, есть ли в строке ошибки.

При редактировании удаление текста слева от курсора производится клавишей <BackSpace> (←), а справа — клавишей <Delete>. Чтобы копировать, вырезать, удалить текст, его предварительно необходимо выделить. Выделяется текст с помощью нажатой клавиши <Shift> и клавишами со стрелками или нажатой левой кнопкой мыши. Приемы ввода текста мало отличаются от работы в других редакторах. Наиболее часто применяемые команды меню дублируются комбинациями клавиш (табл. 2.1).

Таблица 2.1. *Назначение функциональных клавиш в среде QBASIC*

Клавиши	Назначение
<F1>	Справка по ключевому слову, функции или оператору, на который указывает курсор
<Shift>+<F1>	Вывод на дисплей оглавления справочной информации
<F2>	Вывод на экран списка имен всех задействованных в программе процедур и функций, а также самой программы
<Sfift>+<F2>	Вывод на экран следующей процедуры или функции
<Ctrl>+<F2>	Вывод на экран предыдущей процедуры или функции
<F3>	Повтор поиска по ключевому слову
<F4>	Переход к экрану вывода и обратно
<F5>	Продолжение работы по программе
<Shift>+<F5>	Запуск программы на выполнение
<F6>	Переброс курсора из окна ввода программы в окно непосредственного счета и обратно
<Shift>+<F6>	Переброс курсора из одного окна редактирования в другое и обратно

Таблица 2.1 *(окончание)*

Клавиши	Назначение
<F7>	Выполнение программы до курсора
<F8>	Пошаговое выполнение программы с заходом в процедуры и функции
<F9>	Установка или снятие контрольной точки в программе
<F10>	Пошаговое выполнение программы без захода в процедуры и функции
<Shift>+клавиши со стрелками	Выделение фрагмента программы
<Shift>+<Del>	Вырезание фрагмента программы
<Ctrl>+<Y>	Вырезание строки программы
<Shift>+<Ins>	Вставка в программу ранее вырезанного фрагмента
<Ctrl>+<Ins>	Копирование выделенного фрагмента программы
<Ctrl>+<Shift>	Русский шрифт (правые), английский (левые)
<Ctrl>+<Break>	Приостановка выполнения программы

Примечание

Комбинация из нескольких клавиш, например <Ctrl>+<Shift>, нажимается так: сначала нажать клавишу <Ctrl>, затем, не отпуская <Ctrl>, нажать клавишу <Shift>.

Среда программирования QBASIC позволяет запустить программу на выполнение. Если появилось сообщение об ошибке, и она понятна, то следует щелкнуть по кнопке **ОК**, при необходимости уточнений выберите кнопку **Справка** или обратитесь к *приложению 2*.

Рассмотрим некоторые приемы, применяемые при отладке программ. Например, если программа при расчетах по формулам дает неправильные значения, то в этом случае могут быть две причины: либо неправильно реализованы формулы, либо не реализовано ветвление и программа считает не по той формуле. Чтобы проверить вторую причину, рекомендуется воспользоваться клавишей <F8>, высвечивающей траекторию движения по программе. При этом строки с оператором `IF...THEN` следует сделать многооператорными, поставив в конце, например, пустой оператор `PRINT`. Если какой-либо оператор мешает разобраться в причинах появления ошибки, то его можно вывести из рассмотрения, не стирая, поставив перед ним оператор `REM`. А чтобы вывести из рассмотрения большой фрагмент программы, рекомендуется использовать оператор `GOTO` с указанием номера строки, куда следует перепрыгнуть. Чтобы разобраться с циклом, например, правильно ли считается сумма, можно в него вставить оператор `PRINT`, поставив за ним `SLEEP`, организующий паузу до нажатия любой клавиши. Впрочем, с целью создания паузы можно применять и просто `SLEEP`. Проверить, работает ли цикл или иной фрагмент программы, можно, использовав оператор `PRINT "Я здесь"`.

Интерфейс среды QBASIC представлен на рис. 2.1.

На рис. 2.1 цифрами обозначены следующие элементы среды QBASIC:

1. *Строка главного меню*. Для входа в главное меню необходимо нажать клавишу <Alt>. Затем нажать подсвеченную букву выбранного пункта меня и выбрать нужный пункт подменю. Если пункт подменю оканчивается троеточием (...), значит, при выборе этого пункта появится диалоговое окно.

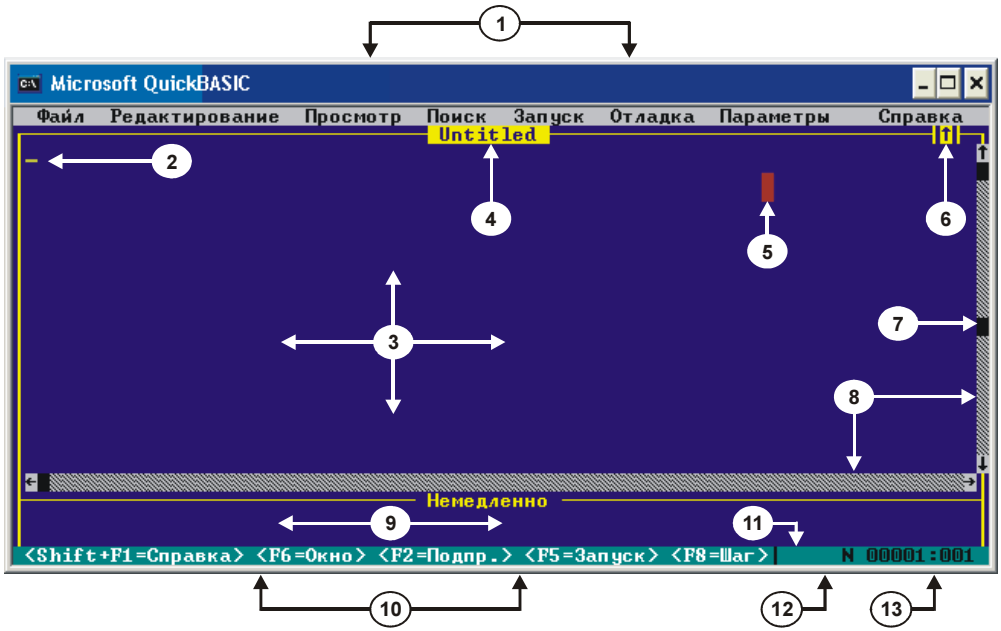


Рис. 2.1. Интерфейс среды QBASIC

2. *Курсор ввода*. Он показывает, в каком месте будет напечатан следующий вводимый символ. Может выглядеть как мигающий символ подчеркивания "_" (режим вставки — вновь вводимые символы раздвигают уже напечатанные) или как мигающий прямоугольник "□" (режим замещения — вновь вводимые символы затирают уже напечатанные).

3. *Рабочее окно*. Именно в рабочем окне вводится и отлаживается текст программы. При разработке большой программы иногда может оказаться полезным работать с двумя частями программы одновременно. Нажмите комбинацию клавиш <Alt>+<M>, <A> (для QBASIC с русифицированным интерфейсом, как на рис. 2.1, используйте клавиши с русскими буквами; <Alt>+<V>, <P> — комбинация клавиш для программы с оригинальным интерфейсом) (меню **Просмотр** | **Разбить** (View | Split)) и рабочее окно разделится на два, в каждом из которых можно работать со своей частью программы. Для перехода между разделенными окнами и окном для немедленного выполнения служат клавиши <F6> и <Shift>+<F6>.

4. *Строка заголовка*. Содержит имя файла (или процедуры), который находится в рабочем окне. Чтобы перейти к другой процедуре или к другому модулю, нажмите клавишу <F2>.

5. *Курсор мыши*. Показывает экранную позицию курсора мыши. С помощью мыши также можно работать с меню. Чтобы открыть меню, подведите курсор мыши к нужному пункту главного меню и нажмите левую кнопку мыши. При работе с текстом программы нажатие правой кнопки мыши на каком-нибудь операторе BASIC вызывает контекстную подсказку об этом операторе (впрочем, гораздо удобнее нажать клавишу <F1>).

6. *Указатель увеличения*. Распахивает рабочее окно на весь экран. Предназначен только для работы с мышью и практического значения не имеет.

7. *Указатель прокрутки*. Показывает относительное место курсора в модуле или процедуре. Если текст модуля или процедуры достаточно длинный, то указатель прокрутки

можно перемещать мышью. Подведите курсор мыши к указателю прокрутки, нажмите правую кнопку мыши и, не отпуская ее, перемещайте курсор по линейке прокрутки вверх или вниз. Аналогичные клавиатурные команды гораздо приятнее — можно просто несколько раз нажать клавишу <PgDn> или <PgUp>, а для перехода в начало текста или процедуры — комбинацию клавиш <Ctrl>+<Home>, в конец — <Ctrl>+<End>.

8. *Линейки прокрутки*. Предназначены для передвижения указателя прокрутки.
9. *Окно для немедленного выполнения (Immediate)*. Для входа в него нажмите клавишу <F6>. Операторы BASIC сразу выполняются после нажатия клавиши <Enter>. Очень удобно использовать при отладке: оборвав выполнение программы, можно изменить значение какой-либо переменной и пустить программу дальше, либо проверить правильность выполнения небольшого участка кода.
10. *Строка контекстных подсказок*. Подсказывает текущие значения функциональных клавиш.
11. *Индикаторы нажатия специальных клавиш*:
    - ^K — появляется, когда устанавливается маркер места;
    - ^P — появляется, когда вводятся специальные управляющие символы.
12. Индикаторы нажатия клавиш-переключателей:
    - C — появляется, когда нажата клавиша <CapsLock>;
    - N — появляется, когда нажата клавиша <NumLock>.
13. *Счетчик строк и столбцов*. Показывает текущую позицию курсора в активном окне.

При работе со средой QBASIC можно использовать мышь. Чтобы указать пункт меню, можно подвести к нему курсор мыши и нажать левую кнопку. Но удобнее использовать клавиатуру. Нажатие клавиши <Alt> и первой буквы меню вызывает его открытие, а для перемещения по диалоговым окнам (например, при загрузке файла или при его сохранении) служат клавиши <Tab> (движение вперед) и <Shift>+<Tab> (движение назад).

Для наиболее часто используемых операций можно применять комбинации клавиш:

- <Atl>+<Ф>, <C> (для англ. интерфейса — <Atl>+<F>, <S>) — сохранение файла программы; осуществляет выбор пункта меню **Файл | Сохранить** (File | Save);
- <Alt>+<Ф>, <O> (<Alt>+<F>, <O>) — открыть файл или документ; осуществляет выбор пункта меню **Файл | Открыть** (File | Open Program);
- <Alt>+<Ф>, <B> (<Alt>+<F>, <X>) — выход из среды QBASIC; осуществляет выбор пункта меню **Файл | Выход** (File | Exit);
- <F1> — получение помощи;
- <F2> — вызвать список процедур и функций;
- <F4> — посмотреть на результат работы программы;
- <Shift>+<F5> — запуск программы на выполнение;
- <Shift>+<F9> — установить/снять точку наблюдения.

Кратко перечислим назначение пунктов меню:

- **Файл** (File) — используется для создания новой программы, загрузки и сохранения программ или частей программ, для печати файлов или частей файлов, для использования команд DOS, выхода из среды QBASIC;

- **Редактирование** (Edit) — для работы с буфером обмена и создания новой процедуры (SUB) или функции (FUNCTION);
- **Просмотр** (View) — для просмотра процедур (SUB) и функций (FUNCTION), включаемых (INCLUDE) файлов, экрана программы; для просмотра процедур (SUB) и функций (FUNCTION);
- **Поиск** (Search) — предназначен для поиска или замены названий переменных, меток или фрагментов исходного текста в активном окне, в текущем модуле или во всех загруженных модулях;
- **Запуск** (Run) — для исполнения загруженной программы, продолжения выполнения прерванной программы, очистки переменных в памяти перед выполнением, создания исполняемого (EXE) файла, определения главного модуля в многомодульной программе;
- **Отладка** (Debug) — используется для отладки программы путем открытия окон наблюдения, которые показывают, как переменные изменяются при работе программы, установки точек прерывания, которые останавливают выполнение программы для того, чтобы вы могли просмотреть значения переменных. Сам термин "debug" в дословном переводе означает "обезжучивание". Он пришел со времен ламповых ЭВМ, когда инженеры не могли отладить программы из-за того, что какой-то таракан забрался в шкаф с электронной начинкой машины и замкнул выводы одной из радиоламп;
- **Параметры** (Options) — для настройки цветов экрана, изменения расположения файла справки и проверки синтаксиса;
- **Справка** (Help) — используется для получения справки по ключевым словам языка BASIC, информации по языку программирования QuickBASIC, контекстно-зависимой помощи, основанной на месторасположении курсора, дополнительных инструкций по получению помощи.

На примере простой программы познакомимся с работой среды QBASIC. Встроенный редактор среды QBASIC обладает всеми основными свойствами редактора текстов и, кроме того, является "интеллектуальным" редактором, т. к. проверяет синтаксис каждой строки текста после того, как вы ее введете. Если синтаксис верен, то строка переводится в псевдокод сразу же, в противном случае появляется описание возникшей ошибки. Редактор также переводит ключевые слова в заглавные буквы и исправляет некоторые ошибки и опечатки. Например, если при вводе строковой константы в операторе PRINT была пропущена вторая кавычка, то редактор автоматически ее поставит.

Введите следующий текст. Регистр букв, в котором набирается программа, значения не имеет. После каждой строки нажимайте клавишу <Enter>.

```
CLS
LOCATE 20, 1
PRINT "Добро пожаловать"
```

В первой строке производится очистка экрана оператором CLS (сокращение от слов *"clear screen"* — очистить экран).

Во второй строке курсор устанавливается на 20 строку, 1 столбец экрана (от слова *"locate"* — разместить; в текстовом режиме экран делится на 25 строк, а каждая строка — на 80 столбцов).

В третьей строке происходит вывод на экран сообщения "Добро пожаловать" с помощью оператора PRINT (от слова *"print"* — напечатать). Обратите внимание, что оператор PRINT произ-

водит вывод только на экран. Чтобы вывести это сообщение на принтер, нужно заменить оператор `PRINT` на `LPRINT`.

Запустим программу на выполнение. Это можно сделать двумя способами:

1. При помощи линейки главного меню. Для этого нажмите комбинацию клавиш <Alt>+<З> (<Alt>+<R>). Будет выделено меню **Запуск** (Run). С помощью клавиши со стрелкой вниз подведите выделенную строку к пункту **Запуск** (Start) и нажмите клавишу <Enter> (рис. 2.2).

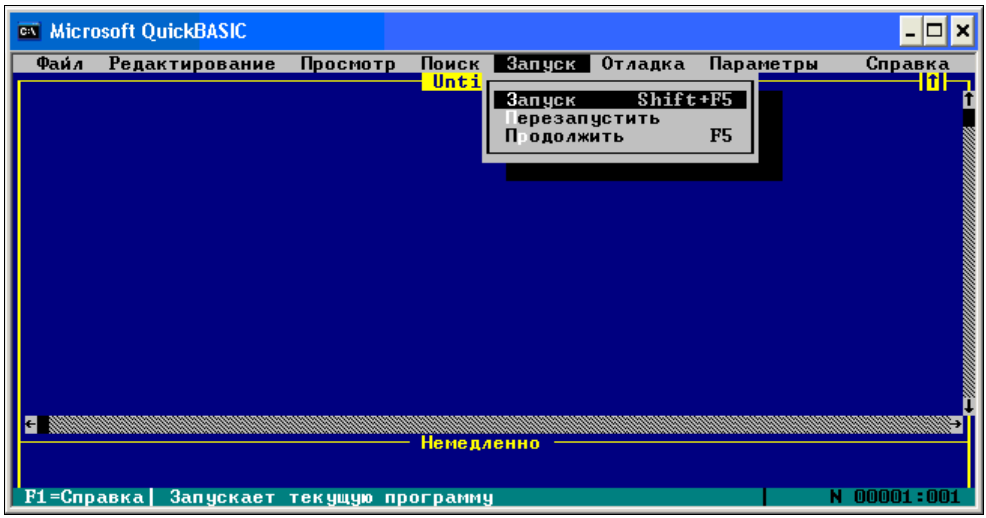


Рис. 2.2. Запуск программы на выполнение с помощью главного меню

2. При помощи клавиатурных сокращений. Для этого нажмите комбинацию клавиш <Shift>+<F5>.

В результате выполнения этой простой программы в нижней части экрана будет напечатано:

```
Добро пожаловать.
Чтобы продолжить, нажмите любую клавишу
(Press any key to continue)
```

С первым сообщением все понятно — в третьей строке программы было сказано напечатать это сообщение. Второе сообщение выводит среда QBASIC. Оно означает "Нажмите любую кнопку для продолжения". Нажмите пробел — на экране снова вернулся текст нашей программы в среде выполнения. Можно продолжить работу над текстом. Чтобы снова посмотреть на экран программы, можно нажать клавишу <F4> или выбрать пункт **Просмотр | Экран вывода** (View | Output Screen).

Теперь эту программу нужно сохранить. Для этого выберите пункт меню **Файл | Сохранить** (File | Save) (рис. 2.3). Вообще, надо взять себе за правило регулярно записывать программу на диск. Рекомендуется это делать при каждом запуске программы. При каких-либо сбоях в работе программы ее исходный текст будет сохранен. Не забывайте также по окончании работы сохранять копию вашей программы на любом внешнем носителе (флеш-карте или диске). Как говорил в таких случаях Peter Norton: "Backup often!" ("Чаще делайте резервные копии").

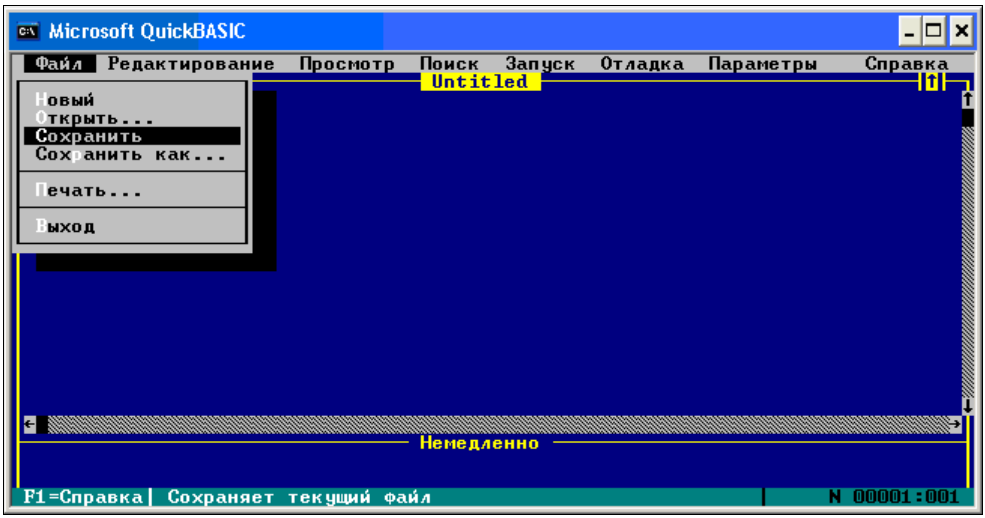


Рис. 2.3. Сохранение программы с помощью главного меню

Для ускорения работы можно нажать комбинацию клавиш <Alt>+<Ф>, <C> (<Alt>+ +<F>, <S>). К счастью, на все часто используемые операции в BASIC предусмотрены быстрые клавиши. И в дальнейшем, описывая действия с линейкой главного меню, будем опираться именно на них.

Мы не присвоили имя нашей программе, поэтому сейчас самое время это сделать. На экране появится окно для ввода имени файла, в котором его можно сохранить (рис. 2.4).

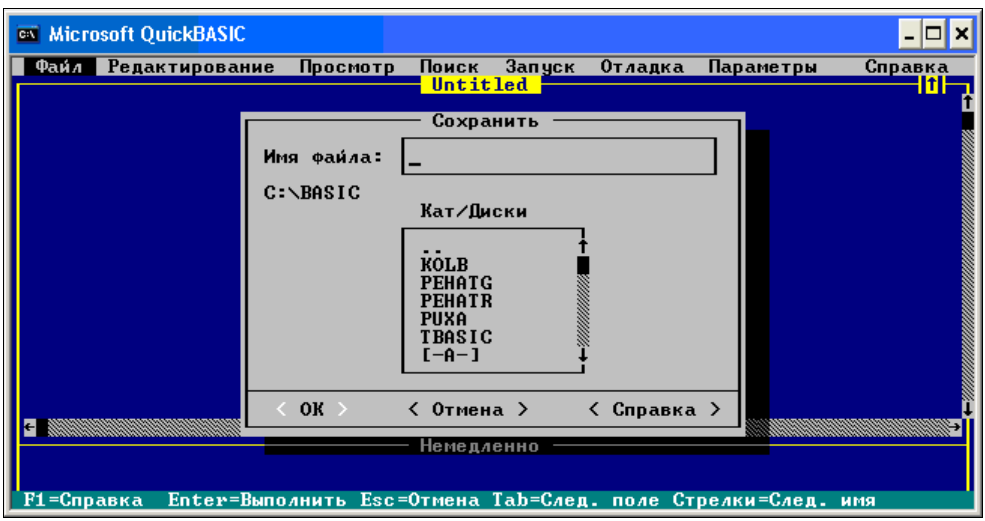


Рис. 2.4. Задание имени файла и формата программы

Введите имя файла: TEST (расширение bas вводить не обязательно, QBASIC добавит его сам). Нажмите клавишу <Tab>. Курсор переместится к списку доступных дисков. Для подтверждения записи нажмите клавишу <Enter>.

Программа сохранена. Ее название TEST.BAS появилось под линейкой главного меню, где раньше было написано "Untitled" (без названия).

Как уже упоминалось, интеллектуальный редактор среды QBASIC осуществляет синтаксический контроль вводимого текста. Если строка написана правильно, то после того, как вы перевели курсор на следующую строку, все ключевые слова BASIC будут написаны заглавными буквами, операторы отделены друг от друга. Это позволяет сразу понять, есть ли в строке ошибки.

Например, вы набираете текст (этот небольшой код выводит ASCII-коды символов от 32 до 255).

```
for i=32 to 255:print chrS(i);:next i
```

После перевода курсора на следующую строку или при запуске этой программы строка примет вид:

```
FOR i = 32 TO 255: PRINT CHRS(i); : NEXT i
```

Если же случайно допущена ошибка с точки зрения синтаксиса языка BASIC, то будет высвечено диалоговое окно с описанием возникшей ошибки, а курсор установится на место предполагаемой ошибки. Так, на рис. 2.5 показана ошибка, возникающая, когда пропущено ключевое слово `THEN` в конструкции `IF...THEN`.

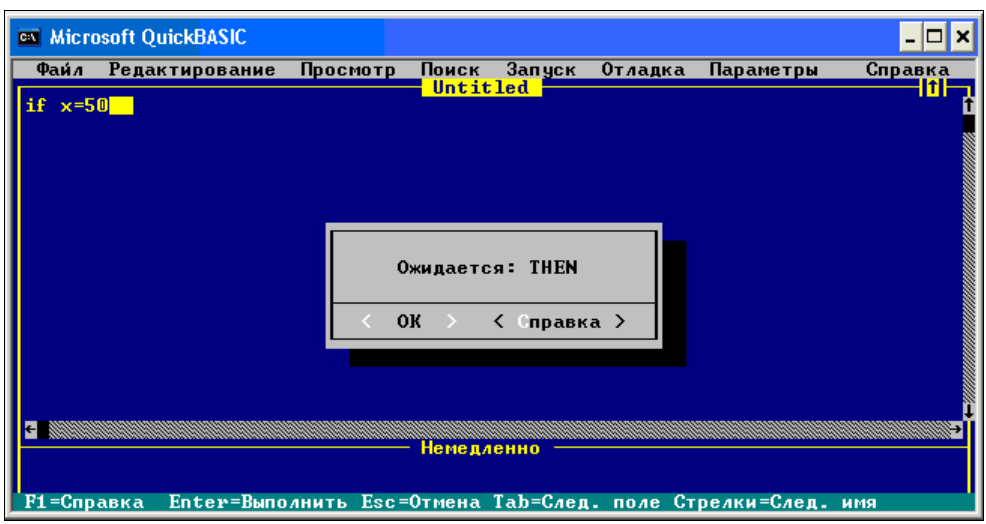


Рис. 2.5. Пример ошибки

Если ошибка понятна без разъяснений, можно нажать клавишу <Enter>. При необходимости уточнений с помощью клавиши <Tab> выберите пункт **Справка** (Help).

Кроме синтаксической поддержки редактор QBASIC поддерживает работу с блоками текста. Это бывает полезно, например, при копировании участков программы между различными модулями или процедурами. Для выделения блоков текста используются следующие сочетания клавиши:

- <Shift>+клавиша со стрелкой влево или со стрелкой вправо — будет выделен строчный фрагмент. Для этого нажмите клавишу <Shift>, и, не отпуская ее, клавишу со стрелкой вправо или влево;

- <Shift>+клавиша со стрелкой вверх или со стрелкой вниз — для выделения прямоугольного фрагмента. Заметьте, что строки могут входить в фрагменты только целиком.

Выделенный фрагмент можно:

- <Delete> — стереть "навсегда";
- <Shift>+<Delete> — стереть, но запомнить в буфере (Clipboard);
- <Ctrl>+<Insert> — запомнить в буфере, не стирая;
- <Shift>+<Insert> — вставить текст из буфера.

Текст, запомненный в буфере, можно вставить в другое место программы неограниченное количество раз.

К числу несомненных достоинств среды QBASIC относится и система оперативной подсказки. Чтобы вы ни делали, нажав клавишу <F1>, всегда можно получить исчерпывающее объяснение. Но можно и самому найти нужный раздел — нажмите <Alt>+<C>, <C> (для англ. интерфейса <Alt>+<H>, <C>) для входа в главное меню оперативной подсказки (рис. 2.6).

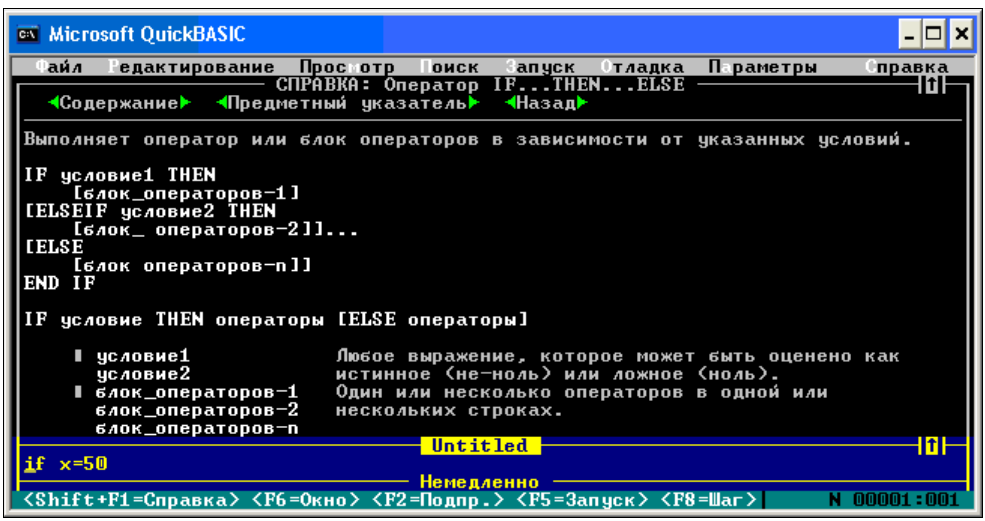


Рис. 2.6. Оперативная подсказка

Здесь можно получить помощь по всем разделам BASIC. Для этого подведите курсор к названию нужной темы и нажмите клавишу <Enter>. Для получения помощи по всем ключевым словам BASIC нажмите <Alt>+<C>, <П> (<Alt>+<H>, <I>). Если вы подведете курсор к наименованию функции или процедуры и нажмете клавишу <F1>, то получите параметры вызова, к имени переменной — ее тип, в каком месте программы она используется. Но самым важным свойством оперативной подсказки считается возможность получения помощи по всем ключевым словам BASIC непосредственно из программы. Просто подведите курсор к нужному слову и нажмите клавишу <F1>. Пример подсказки для оператора `PRINT` приведен на рис. 2.7.

Можно уточнить синтаксис оператора, детали использования, и даже просмотреть пример и скопировать его в вашу программу, модифицировать под собственные нужды.

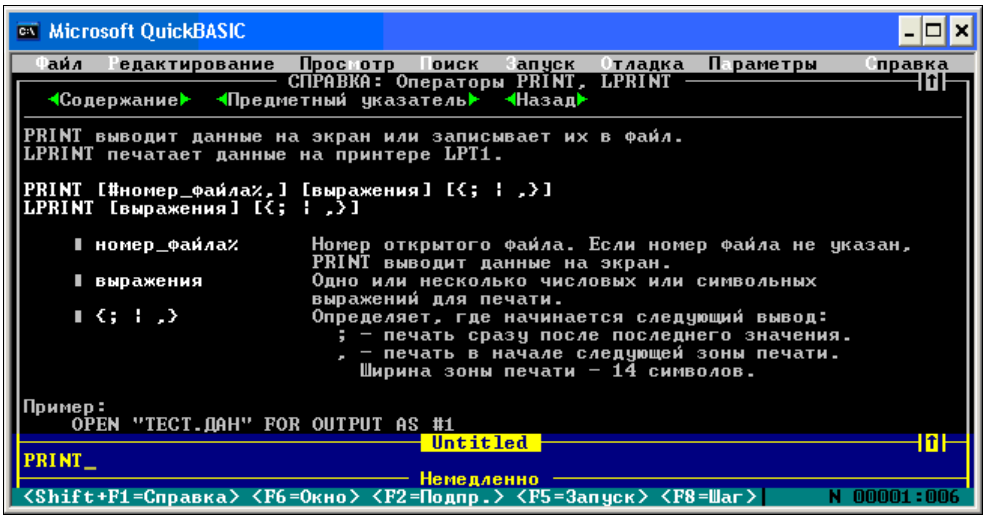


Рис. 2.7. Пример подсказки для оператора `PRINT`

2.2. Пункты меню

Главное меню включает в себя пункты **Файл** (File), **Редактирование** (Edit), **Просмотр** (View), **Поиск** (Search), **Запуск** (Run), **Отладка** (Debug), **Параметры** (Options), **Справка** (Help). Рассмотрим подробнее содержание пунктов меню.

2.2.1. Меню *Файл* (*File*)

Меню **Файл** (File) (рис. 2.8) используется для:

- создания новой программы;
- загрузки и сохранения программ;
- печати файлов или частей файлов;
- выхода из QBASIC.

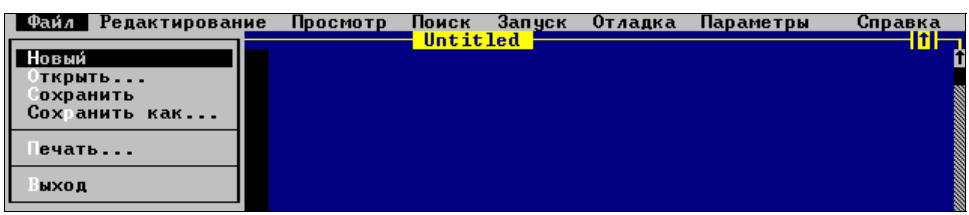


Рис. 2.8. Меню **Файл** (File)

Пункт меню *Новый* (*New*)

Используется для очистки памяти перед началом новой программы.

Пункт меню *Открыть* (*Open*)

Используется для очистки памяти и загрузки сохраненной ранее программы с диска, после чего ее можно запускать или вносить изменения (рис. 2.9).

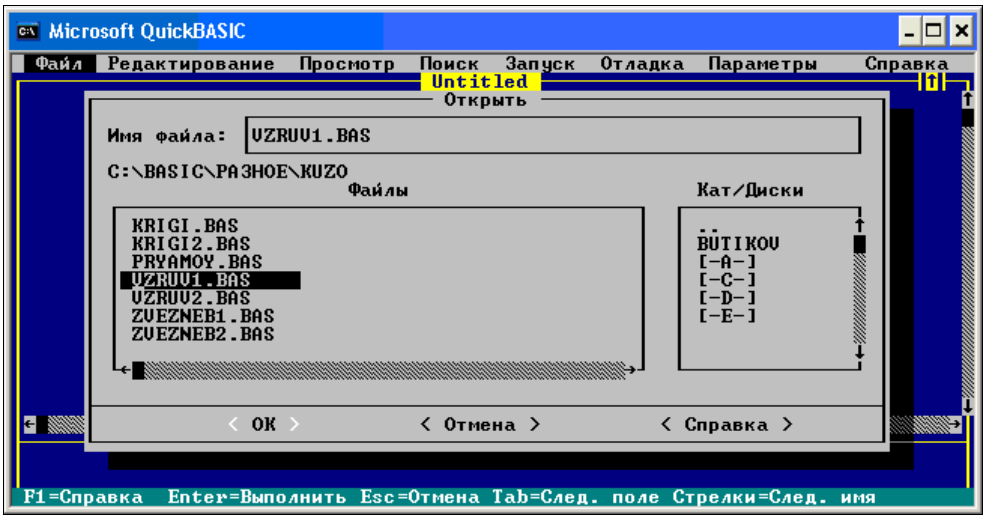


Рис. 2.9. Открытие программы

Пункт меню *Сохранить* (*Save*)

Используется для записи содержимого файла, находящегося в рабочем окне, на диск. Если файл не имеет имени, то будет запрошено его имя и формат. В случае если на диске уже существует файл с таким именем, то он будет перезаписан.

Пункт меню *Сохранить как* (*Save As*)

Используется для записи файла в рабочем окне на диск (рис. 2.10). Файл остается в памяти. С помощью этого пункта меню вы можете сделать несколько копий файла с различными изменениями, сохранив при этом оригинальную версию.

Пункт меню *Печать* (*Print*)

Используется для распечатки текста, находящегося в памяти (рис. 2.11).

Вы можете распечатать:

- выделенный текст;
- содержание активного окна;
- текущий модуль (включая процедуры);
- все модули в многомодульной программе.

Текст посылается на печать через порт DOS LPT 1.

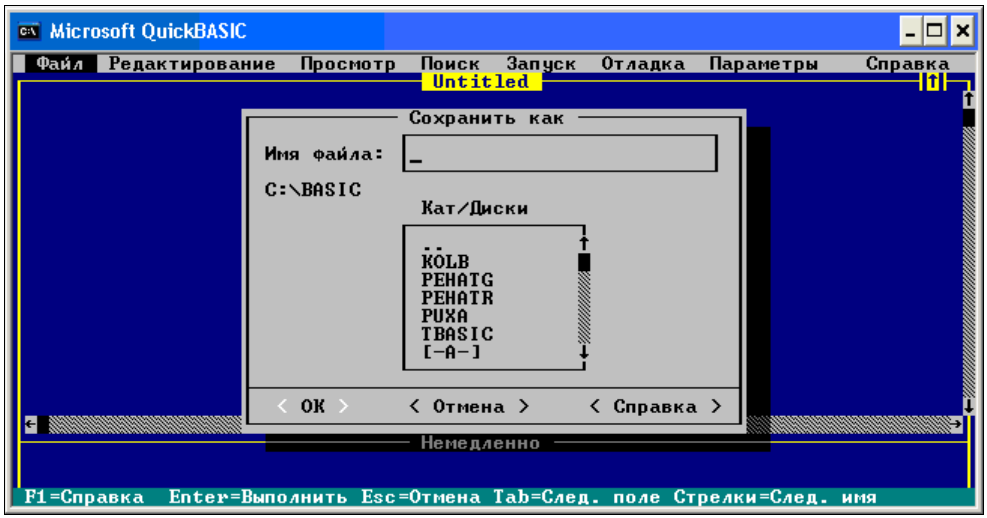


Рис. 2.10. Запись файла на диск с помощью пункта меню **Сохранить как**

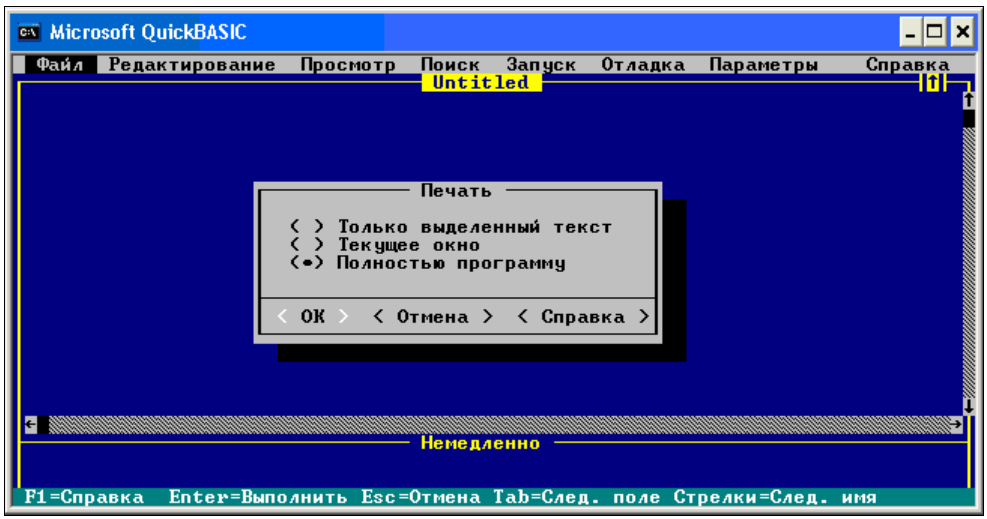


Рис. 2.11. Диалоговое окно **Печать** (Print)

Пункт меню *Выход* (*Exit*)

Используется для выхода из среды QuickBASIC. Если файл не назван, либо не сохранены последние изменения, то может появиться диалоговое окно, в котором нужно выбрать:

- **<Да>** (<Yes>) для записи изменений;
- **<Нет>** (<No>) для отмены изменений;
- **<Отмена>** (<Cancel>) для отмены выхода.

2.2.2. Меню *Редактирование* (*Edit*)

Меню **Редактирование** (Edit) (рис. 2.12) используется для:

- стирания (или удаления в буфер) текста;
- копирования текста;
- передвижения (удаления и вставки через буфер) текста;
- стирания текста без запоминания в буфер;
- создания новой процедуры (SUB) или функции (FUNCTION)

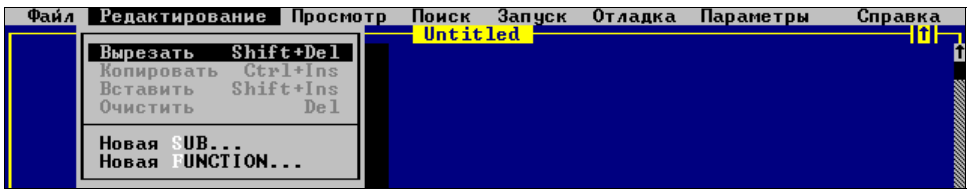


Рис. 2.12. Меню **Редактирование** (Edit)

Пункт меню *Отменить* (*Undo*)

Используется для отмены всех изменений в текущей строке, пока курсор находится на ней. Для отмены изменений возможно применение комбинации клавиш <Alt>+<BackSpace>. Однако следует учитывать, что отмена изменений возможна до тех пор, пока курсор находится на текущей строке.

Пункт меню *Вырезать* (*Cut*)

Используйте пункт меню **Вырезать** (Cut) или нажмите комбинацию клавиш <Shift>+ +<Delete> для удаления выбранного текста из активного окна и помещения его в буфер.

Затем вы можете использовать пункт меню **Вставить** (Paste) или комбинацию клавиш <Shift>+<Insert>, чтобы вставить текст в активное окно. Чтобы начать выделение текста, поставьте курсор в нужное место и нажимайте клавишу <Shift> и нужную клавишу со стрелкой. Для стирания текста без запоминания его в буфере нажмите клавишу <Delete>. Вырезанный текст остается в буфере и может быть вставлен в любое место программы.

Пункт меню *Копировать* (*Copy*)

Используйте пункт меню **Копировать** (Copy) или нажмите комбинацию клавиш <Ctrl>+<Insert> для копирования выбранного текста из активного окна в буфер. Оригинальный блок текста остается без изменения.

Затем вы можете использовать пункт меню **Вставить** (Paste) или нажать комбинацию клавиш <Shift>+<Insert>, чтобы вставить текст в активное окно. Вырезанный текст остается в буфере и может быть скопирован в другое место программы.

Пример в табл. 2.2 иллюстрирует действие команд **Вырезать**—**Вставить** и **Копировать**—**Вставить**.

Таблица 2.2. *Действие команд* ***Вырезать—Вставить*** *и* ***Копировать—Вставить***

	Вырезать (<Shift>+<Delete>)		**Копировать (<Ctrl>+<Insert>)**	
	Текст	**Буфер**	**Текст**	**Буфер**
Выделить текст (<Shift>+клавиши со стрелками)	*текст*		*текст*	
Выбрать соответствующий пункт меню или нажать комбинацию клавиш		*текст*	*текст*	*текст*
Переместить курсор и выбрать пункт меню **Вставить** (Paste) (<Shift>+<Insert>)		*текст*	*текст*	*текст*

Пункт меню *Вставить* (*Paste*)

Используйте пункт меню **Вставить** (Paste) или нажмите комбинацию клавиш <Shift>+ +<Insert>, чтобы скопировать блок текста из буфера в любое место активного окна.

Позиция, куда будет вставлен текст, определяется:

- выделенным текстом. Он стирается, и текст из буфера копируется на его место;
- позицией курсора, если нет выделенного текста. Тогда:
  - если текст в буфере меньше, чем одна строка текста, он копируется слева от курсора;
  - если текст в буфере больше, чем одна строка, он копируется выше курсора.

Пункт меню *Очистить* (*Clear*)

Используйте пункт меню **Очистить** (Clear) или нажмите клавишу <Delete>, чтобы стереть выделенный текст из активного окна без запоминания в буфере. Содержимое буфера остается без изменений.

Пункт меню *Новая SUB* (*New SUB*)

Используйте пункт меню **Новая SUB** (New SUB), чтобы создать новую процедуру как часть программы или модуля в рабочем окне. Появится окно диалога для ввода названия процедуры (рис. 2.13). BASIC очистит рабочее окно и сгенерирует операторы `SUB` и `END SUB`, так что вы сможете начать вводить текст. Процедура может быть определена только один раз во всей программе.

Когда вы вводите текст внутри новой программы, используйте пункт меню **Просмотр | SUBs** (View | SUBs) или нажмите клавишу <F2>, чтобы вернуться в программу. Ваша процедура появится в списке процедур пункта меню **SUBs**.

Вы можете начать вводить процедуру и без использования меню. Для этого введите ключевое слово BASIC `SUB` и имя процедуры.

Пункт меню *Новая FUNCTION* (*New FUNCTION*)

Используется для ввода определения функции в рабочем окне. Функция в BASIC — это процедура, которая возвращает значение в вашу программу. Каждая функция может быть определена только один раз во всей программе.

Диалоговое окно **Новая FUNCTION** (New FUNCTION) аналогично окну на рис. 2.13.

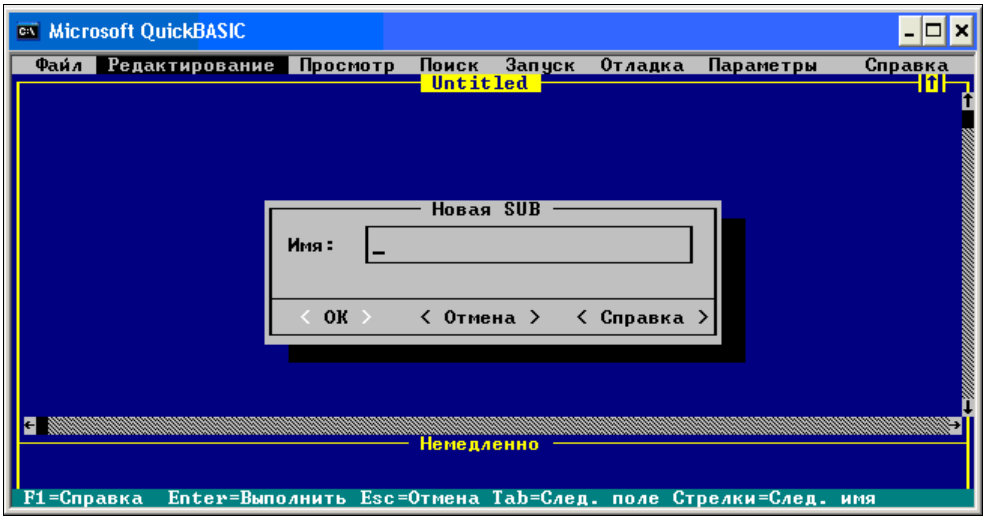


Рис. 2.13. Диалоговое окно **Новая SUB** (New SUB)

2.2.3. Меню *Просмотр* (*View*)

Меню **Просмотр** (View) используется для:

- просмотра процедур (`SUB`) и функций (`FUNCTION`);
- просмотра выходного экрана.

Рис. 2.14. Меню **Просмотр** (View)

Пункт меню *SUBs*

Используется для просмотра различных частей загруженной программы (рис. 2.15).

Для вызова этого пункта меню можно воспользоваться клавишей <F2>. Вы сможете:

- просмотреть содержимое рабочего окна;
- разделить рабочее окно для просмотра различных частей одной программы или двух программ одновременно;
- стереть процедуру или модуль.

Пункт меню *Разбить* (*Split*)

Используется для работы с двумя частями программы или двумя программами одновременно. Выбор этого пункта делит рабочее окно горизонтально. Используйте клавиши <F2> и <Shift>+<F6> для передвижения между окнами на экране:

- <F6> — передвигает курсор в нижнее окно;
- <Shift>+<F6> — передвигает курсор в верхнее окно.

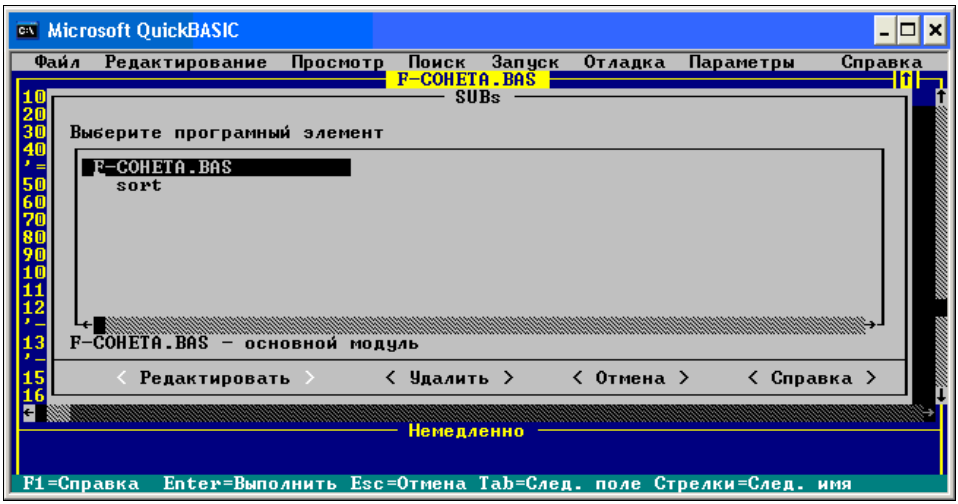


Рис. 2.15. Диалоговое окно **SUBs**

Для того чтобы восстановить первоначальное состояние окна, выберите пункт меню **Разбить** (Split) снова.

Когда активное окно содержит курсор, вы можете управлять размерами этого окна с помощью клавиатуры:

- <Alt>+<+> (плюс) — увеличивает размер окна на одну строку;
- <Alt>+<–> (минус) — уменьшает размер окна на одну строку;
- <Ctrl>+<F10> — распахивает окно на весь экран или возвращает его первоначальный размер.

Пункт меню *Экран вывода* (*Output Screen*)

Используется для переключения между средой BASIC и выходным экраном вашей программы. Также можно пользоваться клавишей <F4>.

2.2.4. Меню *Поиск* (*Search*)

Меню **Поиск** (Search) (рис. 2.16) используется для поиска или замены названий переменных, меток или фрагментов исходного текста.

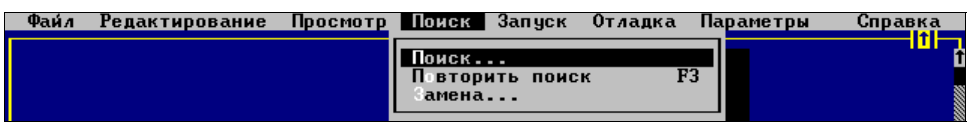


Рис. 2.16. Меню **Поиск** (Search)

Пункт меню *Поиск* (*Find*)

Используется для поиска текстовой строки (рис. 2.17):

- в активном окне;
- в текущем модуле;
- во всех загруженных модулях.

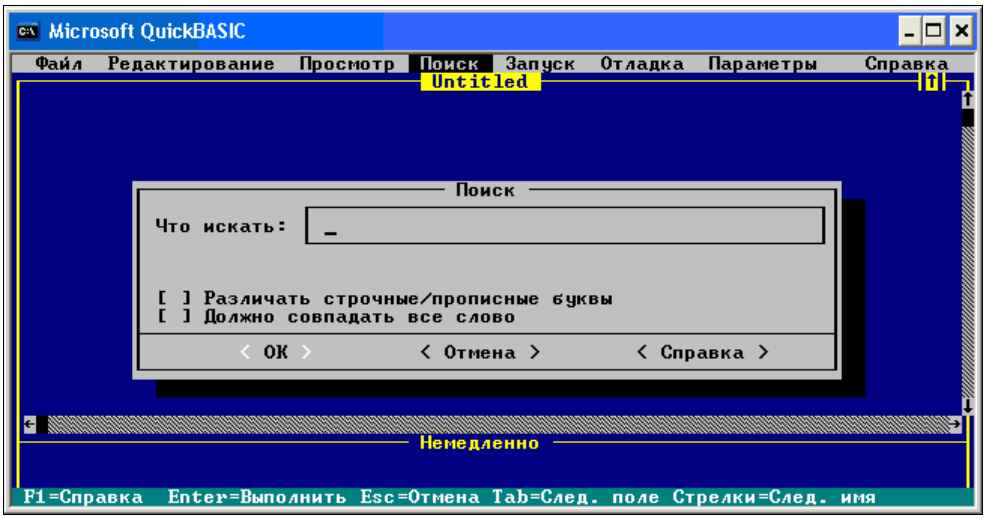


Рис. 2.17. Диалоговое окно **Поиск** (Find)

Вы можете выбрать как точный образец (с учетом регистра), так и целое слово. Для того чтобы заменить строку, используйте пункт меню **Замена** (Change).

Пункт меню *Повторить поиск* (*Repeat Last Find*)

Используется для повторения последнего поиска, выполненного ранее командой **Поиск** (Find) или **Замена** (Change). Вместо вызова пункта меню может быть нажата клавиша <F3>.

Если с момента запуска BASIC команды **Поиск** (Find) или **Замена** (Change) не выполнялись, то будет найдено:

- слово, на которое указывает курсор;
- если курсор не указывает на слово, то будет найдено слово слева от курсора.

Пункт меню *Замена* (*Change*)

Используется для поиска текстовой строки и замены ее на другую (рис. 2.18).

Можно выбрать, необходимо ли вам:

- поиск с учетом регистра или нет;
- показ места, где будет произведена замена;

2.2.5. Меню *Запуск* (*Run*)

Меню **Запуск** (Run) (рис. 2.19) используется для:

- исполнения загруженной программы;
- продолжения выполнения прерванной программы;
- очистки переменных в памяти перед выполнением.

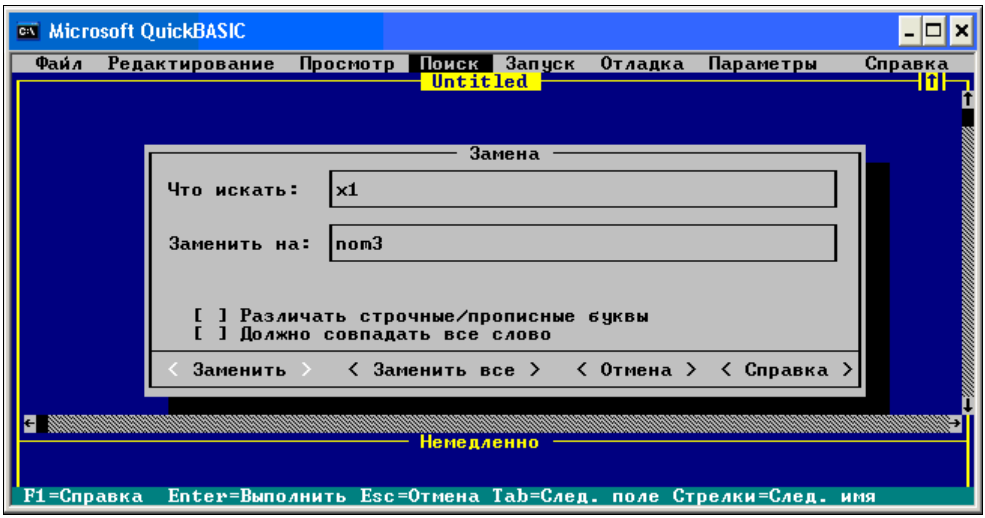


Рис. 2.18. Диалоговое окно **Замена** (Change)

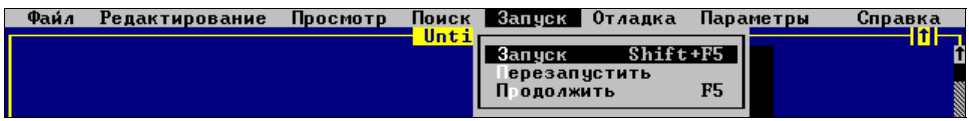


Рис. 2.19. Меню **Запуск** (Run)

Пункт меню *Запуск* (*Start*)

Используется для очистки памяти и запуска программы на исполнение, начиная с первого исполняемого оператора в главном модуле.

Команда **Запуск** (Start) может использоваться после остановки программы нажатием комбинации клавиш <Ctrl>+<Break> для того, чтобы запустить программу с начала. Для исполнения команды **Запуск** (Start) можно нажать комбинацию клавиш <Shift>+<F5>. Вместо <Ctrl>+<Break> можно пользоваться точками прерывания и точками наблюдения (см. меню **Отладка** (Debug)).

Пункт меню *Перезапуск* (*Restart*)

Пункт меню **Перезапуск** (Restart) очищает память и продолжает выполнение первого исполняемого оператора. Используйте клавишу <F8> для выполнения одного оператора. Вы можете переопределить, какой оператор будет выполняться следующим с помощью меню **Отладка | Установить следующее выражение** (Debug | Set Next Statement).

Пункт меню *Продолжить* (*Continue*)

Используйте пункт меню **Продолжить** (Continue) после того, как программа была остановлена, чтобы продолжить ее выполнение.

Эта команда часто используется для продолжения выполнения программы, остановленной на точке прерывания или точке наблюдения, после того, как вы проверили значения переменных или посмотрели выходной экран.

Для выполнения команды **Продолжить** (Continue) можно также нажать клавишу <F5>.

2.2.6. Меню *Отладка* (*Debug*)

Меню **Отладка** (Debug) (рис. 2.20) используется для:

- открытия окон наблюдения, которые показывают, как переменные изменяются при работе программы;
- установки точек прерывания, прерывающих выполнение программы с целью просмотра значений переменных.

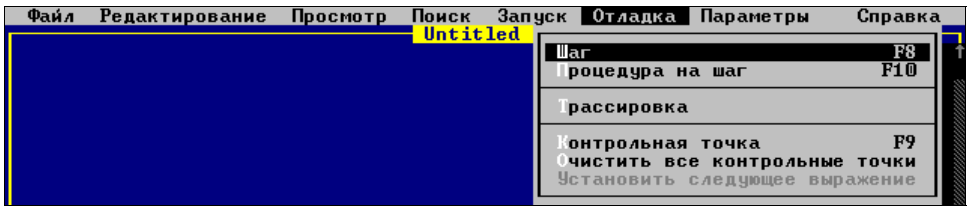


Рис. 2.20. Меню **Отладка** (Debug)

Пункт меню *Шаг* (*Step*)

Используйте пункт меню **Шаг** (Step) (или нажмите клавишу <F8>) для выполнения одного выражения в вашей программе. Эта команда позволяет вам выполнять программу, включая процедуры, шаг за шагом.

Пункт меню *Процедура на шаг*

Используйте **Процедура на шаг** (или нажмите клавишу <F10>) для выполнения одного выражения в вашей программе или для выполнения одной процедуры, при этом вызов процедуры рассматривается как один шаг. Команда **Процедура на шаг**, как и команда **Шаг** (Step), позволяет вам проводить пошаговую отладку программы, но сокращает время отладки, пропуская процедуры, для которых известно, что они работают правильно.

Пункта меню **Процедура на шаг** нет в английском варианте программы.

Пункт меню *Трассировка* (*Trace On*)

Используется для выделения каждого выполняемого в текущий момент времени выражения в вашей программе. Это позволяет вам проследить выполнение программы.

Пункт меню *Контрольная точка* (*Watchpoint*)

Используйте команду **Контрольная точка** (Watchpoint) (или нажмите клавишу <F9>), чтобы включать или выключать контрольные точки. Контрольные точки — это маркеры, установленные на выражениях в вашей программе. Если вы выполняете программу, и она встречает контрольную точку, программа прерывается на этом выражении. BASIC выделяет подсветкой строку контрольной точки. Чтобы включить или выключить контрольную точку:

1. Установите курсор в строке, в которой вы хотите включить контрольную точку, или выделите подсветкой строку контрольной точки.
2. Выберите пункт меню **Контрольная точка** (Watchpoint) (или нажмите клавишу <F9>).

Используйте контрольную точку, чтобы приостановить вашу программу в том месте, где вы ожидаете трудность, чтобы затем:

- вывести значение переменных в окне **Немедленно** (Immediate);
- продолжить выполнение программы после контрольной точки пошагово.

Пункт меню *Очистить все контрольные точки* (*Clear All Breakpoints*)

Используется для удаления всех контрольных точек из вашей программы. Используйте команду **Контрольная точка** (Watchpoint) (или нажмите клавишу <F9>), чтобы выключить отдельные контрольные точки.

2.2.7. Меню *Параметры* (*Options*)

Меню **Параметры** (Options) (рис. 2.21) используется для:

- настройки цветов экрана;
- установки путей для поиска служебных файлов;
- включения/выключения режима **Проверка синтаксиса** (Syntax Checking).

В английском варианте программы есть также пункты меню для включения/выключения режима полного меню (**Full Menus**) и для переопределения правой кнопки мыши (**Right Mouse**).

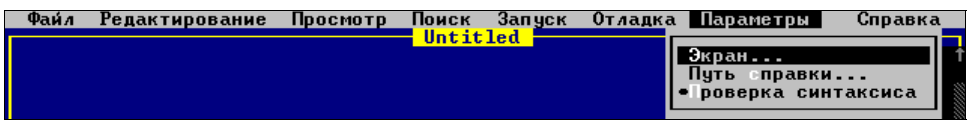


Рис. 2.21. Меню **Параметры** (Options)

Сделанные вами установки автоматически запоминаются и используются при дальнейших вызовах BASIC.

Пункт меню *Экран* (*Display*)

Пункт меню **Экран** (Display) (рис. 2.22) используется для установки:

- экранных цветов;
- линеек прокрутки в окнах и списках;
- числа пробелов, выдаваемых клавишей <Tab>.

Пункт меню *Путь справки* (*Set Paths*)

Используется для замены каталогов, которые просматривает BASIC при поиске файла справки QBASIC.HLP (рис. 2.23).

Пункт меню *Проверка синтаксиса* (*Syntax Checking*)

Эта команда включает/выключает Интеллектуальный Редактор BASIC, который:

- проверяет строку на наличие синтаксических ошибок;
- форматирует строку;

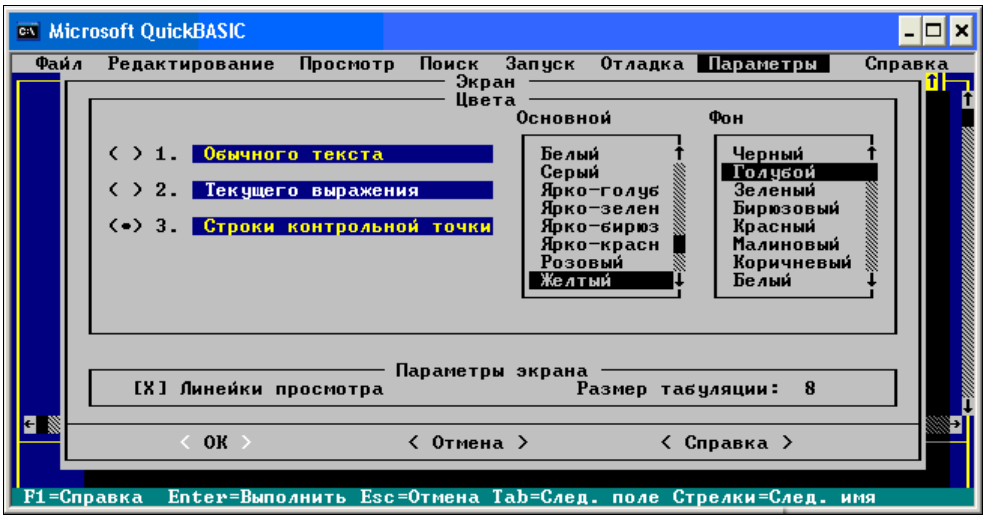


Рис. 2.22. Диалоговое окно **Экран** (Display)

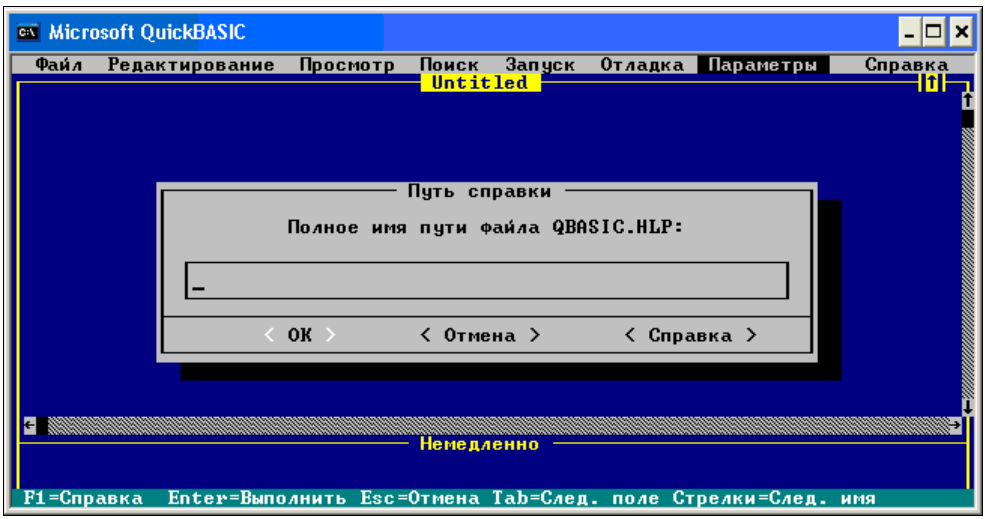


Рис. 2.23. Диалоговое окно **Путь справки** (Set Paths)

- в случае если синтаксис в строке правильный, то переводит строку в исполнимую форму, при этом операторы языка BASIC становятся написанными заглавными буквами.

Выключение этой команды отменяет эти функции и превращает Интеллектуальный Редактор в обычный текстовый. Когда **Проверка синтаксиса** (Syntax Checking) включена, то рядом с этим пунктом меню появляется точка.

Интеллектуальный Редактор действует в режимах:

- **Файл** | **Новый** (File | New Program) или **Файл** | **Открыть** (File | Open Program);
- в английской версии программы если при открытии файла с помощью меню **File** | **Create File** (Создать файл) или **File** | **Load File** (Загрузить файл) не был выбран пункт **Document**

(документ), то Интеллектуальный Редактор отключается, и вы можете использовать BASIC как обычный редактор для ввода текста.

2.2.8. Меню *Справка* (*Help*)

Меню **Справка** (Help) (рис. 2.24) используется для получения:

- справки по ключевым словам BASIC;
- информации по языку программирования BASIC;
- контекстно-зависимой помощи, основанной на месторасположении курсора;
- дополнительных инструкций по получению помощи.

Рис. 2.24. Меню **Справка** (Help)

Пункт меню *Предметный указатель* (*Index*)

Используется для вывода на экран предметного указателя справки, представляющего собой список тем справки в алфавитном порядке, включая ключевые слова языка BASIC (рис. 2.25). Каждый термин в предметном указателе снабжен дополнительной информацией. Чтобы получить справку:

1. Выберите команду **Предметный указатель** (Index) из меню **Справка** (Help).
2. Нажмите клавишу, соответствующую первой букве темы.
3. Установите курсор на теме, по которой вы хотите получить справку, нажмите <F1>.

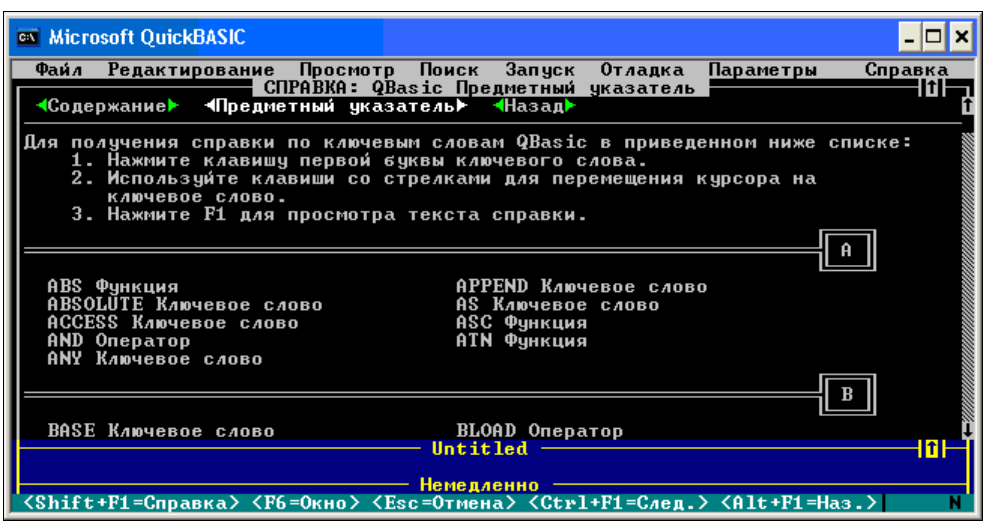


Рис. 2.25. Предметный указатель справки

Пункт меню *Содержание* (*Contents*)

Используется для вывода на экран справочной таблицы содержания (рис. 2.26). Эта таблица поможет вам ориентироваться в справке BASIC. Чтобы получить справку по любой теме в таблице:

1. Выберите команду **Содержание** (Contents) из меню **Справка** (Help).
2. Нажмите клавишу с буквой, соответствующей первой букве темы. Повторяйте, пока курсор не будет находиться на теме, по которой вы хотите получить справку.
3. Нажмите клавишу <F1>.

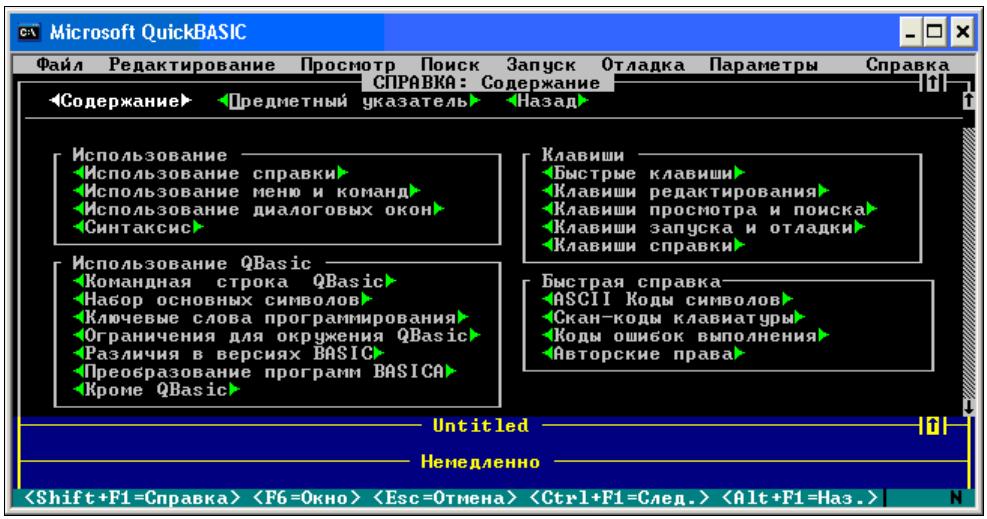
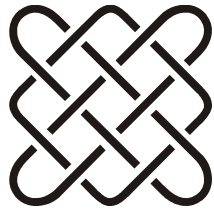


Рис. 2.26. Окно **Содержание** (Contents)

Пункт меню *Использование справки* (*Help on Help*)

Применяется для вывода темы "Использование справки" ("Help on Help"). Использование справки поможет вам в работе с мышью и клавиатурой для получения справки по теме, выведенной на экране BASIC.



ГЛАВА 3

Ввод данных

Ввод данных является начальной стадией работы с программой. Данные могут быть представлены отдельными переменными и константами, массивами и файлами сложной структуры. Вид и структура данных зависит от решаемой задачи. Когда программа разработана, ей необходимо сообщить конкретные значения переменных, которые она должна преобразовать в результат. Бывает, что неудачно выбранная структура входных данных неоправданно усложняет тело программы, и, наоборот, правильно определенная входная последовательность делает программу простой и логичной. В BASIC ввод данных можно осуществить разными способами: с помощью оператора присваивания, посредством оператора `INPUT`, пары операторов `READ/DATA` или из файла. Какой же способ ввода данных выбрать, определяется логикой программы.

3.1. Оператор присваивания

Если переменных и параметров в программе немного, а их значения можно задать раз и навсегда, то удобнее всего ввод данных осуществлять с помощью оператора присваивания.

Например, `a = 2`, где `=` и есть оператор присваивания. В левой части оператора присваивания может находиться только имя той переменной, в которую будет заноситься новое значение (и больше ничего!). В правой части может находиться: конкретное значение (число или символ, или строка — в зависимости от типа переменной), арифметическое или алгебраическое выражение, имя другой переменной, уже имеющей значение. Типы переменных в левой и правой частях оператора присваивания должны совпадать. Например:

```
40 y = 2 * x ^ 2 + 4 * x — SIN (x)
50 z$ = "верно"
```

Результат вычисления выражения, стоящего в правой части оператора присваивания в строке 40, запоминается в ячейки памяти `y`, выделяемые программой специально для этой переменной. В этом случае говорят, что значение выражения присваивается переменной, стоящей слева от знака присваивания.

Поясним на примерах использование оператора присваивания для ввода данных в программах.

Пример 3.1. Значения функции

Требуется найти значения функции $y = 2x^2 + 5x - 1$ для трех значений переменной: $x = 0$, $x = 1$ и $x = 2$.

Программа 3.1

```
'3P-1opV.bas
10 CLS   'очистка экрана
20 PRINT "Найти три значения функции"
30 PRINT "y = 2 * x ^ 2 + 3 * x – 5"
40 PRINT
50 x(1) = 0
60 x(2) = 1
70 x(3) = 2
80 FOR i = 1 TO 3
90 y = 2 * x(i) ^ 2 + 3 * x(i) – 5
100 PRINT TAB(5); "x = "; x(i); TAB(14); "y = "; y
110 NEXT i
120 END
```

Операторы PRINT в строках 20—40, 100 обеспечивают интерфейс программы. Операторы присваивания поставлены в строках 50—70 — для ввода значений аргумента функции и в строке 90 для записи выражения функции, значения которой вычисляются посредством цикла FOR...NEXT в строках 80—110 (строки 90—100 — тело цикла).

Результат работы программы 3.1 приведен на рис. 3.1.

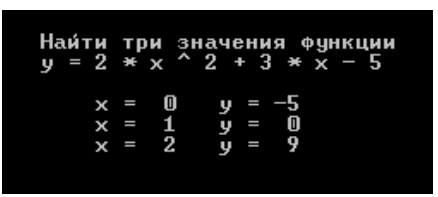


Рис. 3.1. Три значения функции

Оператор присваивания может применяться и для ввода значений символьных (строковых) переменных.

Пример 3.2. Конкатенация символьных переменных

Требуется разработать программу, реализующую слияние символьных переменных.

Программа 3.2

```
'3P-2opV.bas
10 CLS   'очистка экрана
20 PRINT "Конкатенация символьных переменных"
30 PRINT
'==1===== Блок ввода данных ======================================
40 x1$(1) = "папа"
50 x2$(1) = "зол"
60 x1$(2) = "сара"
70 x2$(2) = "фан"
80 x1$(3) = "кара"
90 x2$(3) = "кум"
```

```
'==2===== Блок конкатенации ======================================
100 FOR i = 1 TO 3
110 PRINT "1-е слово — "; x1$(i)
120 PRINT "2-е слово — "; x2$(i)
130 y$ = x1$(i) + x2$(i)
140 PRINT "Конкатенация — "; y$
150 PRINT
160 NEXT i
'================================================================
170 END
```

Программа 3.2 простая и прозрачная и практически не требует комментариев. Ввод данных осуществляется операторами присваивания в строках 40—90, причем тип переменных, как символьных, задается суффиксом — знак доллара ($). Оператор присваивания используется и в строке 130 для записи выражения конкатенации.

Результат работы программы 3.2 дан на рис. 3.2.

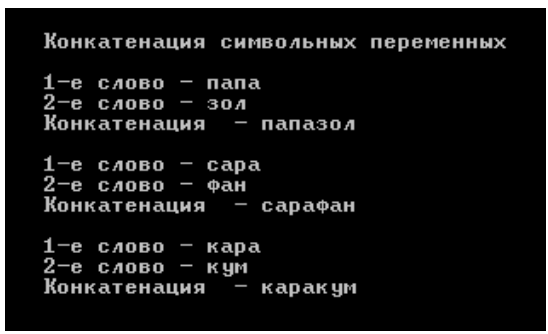


Рис. 3.2. Программа для работы с символьными переменными

В программах, работающих с графикой, с помощью операторов присваивания обычно задаются значения координат начальной или опорной точки. Под опорной понимается такая точка, меняя лишь координаты которой можно переносить все изображение по экрану без изменения.

Пример 3.3. Рисование лесенки

Требуется разработать программу, рисующую лесенку.

Программа 3.3

```
'3P-3opV.bas
10 CLS          'очистка экрана
20 SCREEN 9
30 x = 30
40 y = 50
'======== Блок рисования ========================================
50 LINE (x, y) — STEP(50, 10), 14, BF
60 FOR i = 1 TO 7
```

```
70 LINE STEP(-10, 0) — STEP(50, 10), 14, BF
80 NEXT i
'=============================================================
90 END
```

После установки экранного режима оператором SCREEN в строке 20 посредством оператора присваивания в строках 30—40 задаются координаты опорной точки x = 30 и y = 50. Элементы лесенки рисуются оператором LINE в строке 70, являющимся телом цикла FOR...NEXT в строках 60—80.

Результат работы программы 3.3 показан на рис. 3.3.

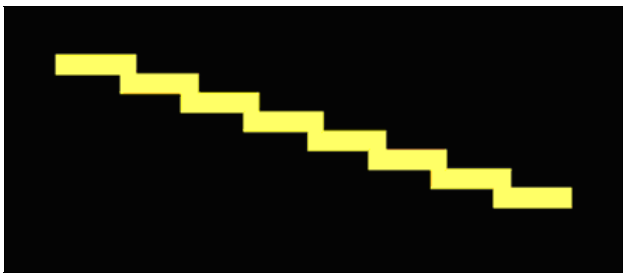


Рис. 3.3. Программа рисования лесенки

3.2. Оператор *INPUT*

Ввод исходных данных может осуществляться и посредством оператора INPUT, обеспечивающего ввод данных с клавиатуры во время выполнения программы и размещение их в списке переменных. Оператор INPUT останавливает программу и ждет ввода данных. Переменные могут быть числовыми, символьными или элементами массивов. Типы и количество данных в списке и вводимых с клавиатуры должны совпадать, например:

```
INPUT "Введите значение x, y ", x, y$
```

В приведенном примере символьная строка, заключенная в кавычки, называется подсказкой или приглашением к вводу и выводится один к одному на экран. Если после приглашения поставить точку с запятой, а не запятую как в примере, то BASIC будет выводить знак вопроса. В данном примере оператор требует ввести численное значение x, завершив его ввод запятой, а затем ввести символьное значение для символьной переменной y$, завершив ввод значений обеих переменных нажатием клавиши <Enter>.

Предпочтительнее же для каждой переменной использовать свой оператор INPUT:

```
10 INPUT "Введите x = ", x
20 INPUT "Введите y = ", y$
```

Или вариант с предваряющим оператором PRINT:

```
10 PRINT "Введите значения x, y"
20 INPUT "x = ", x
30 INPUT "y = ", y$
```

Если введены значения не тех типов или их количество не совпадает с числом переменных в списке INPUT, то появится сообщение об ошибке "Ввод с начала" ("Redo from start"). Если в символьной строке необходимо ввести запятую, то символьную строку нужно заключить

в кавычки. Пока не закончен ввод, значения не присваиваются переменным. До нажатия клавиши <Enter> можно редактировать вводимые данные. Клавиши редактирования, в основном, аналогичны клавишам редактирования среды BASIC (табл. 3.1).

Таблица 3.1. *Клавиши редактирования при вводе данных с помощью оператора* `INPUT`

Клавиши	Выполняемое действие
Клавиша со стрелкой вправо	Курсор на один символ вправо
Клавиша со стрелкой влево	Курсор на один символ влево
<Ctrl>+клавиша со стрелкой вправо	Курсор на одно слово вправо
<Ctrl>+клавиша со стрелкой влево	Курсор на одно слово влево
<Home>	Курсор к началу вводимой строки
<End>	Курсор к концу вводимой строки
<Insert>	Переключение режимов вставка/замещение
<Tab>	Табуляция вправо
<Delete>	Стереть символ под курсором
<BackSpace>	Стереть символ слева от курсора
<Ctrl>+<End>	Стереть от курсора до конца строки
<Esc>	Стереть всю строку ввода
<Enter>	Конец ввода
<Ctrl>+<T>	Включает/выключает список значений F-клавиш в последней строке экрана
<Ctrl>+<Break> или <Ctrl>+<C>	Обрывает выполнение программы

Если оператор присваивания фиксирует значения переменных, и для ввода их новых значений требуется модернизация программы — изменение оператора присваивания, то ввод данных с помощью оператора `INPUT` позволяет естественным образом задавать различные значения переменным при каждом новом запуске программы.

Пример 3.4. Советы постороннего

Требуется разработать программу, дающую совет относительно знаний в зависимости от возраста, с обращением по имени.

Программа 3.4

```
'3I-1opV.bas
10 CLS
20 PRINT "Советы постороннего"
30 PRINT
'==1===== Блок ввода ============================================
40 INPUT "Ваше имя — ", imya$
50 INPUT "Ваш возраст — ", age
```

```
'==2===== Блок анализа ввода ======================================
60 SELECT CASE age
     CASE 0 TO 22
                g$ = "приобретайте и преумножайте знания"
     CASE 23 TO 40
                g$ = "применяйте знания и накапливайте опыт"
     CASE ELSE
                g$ = "передавайте опыт и знания"
70 END SELECT
'==3===== Блок вывода =============================================
80 PRINT
90 PRINT imya$; CHR$(44); g$
'==================================================================
100 END
```

Результат работы программы 3.4 приведен на рис. 3.4.

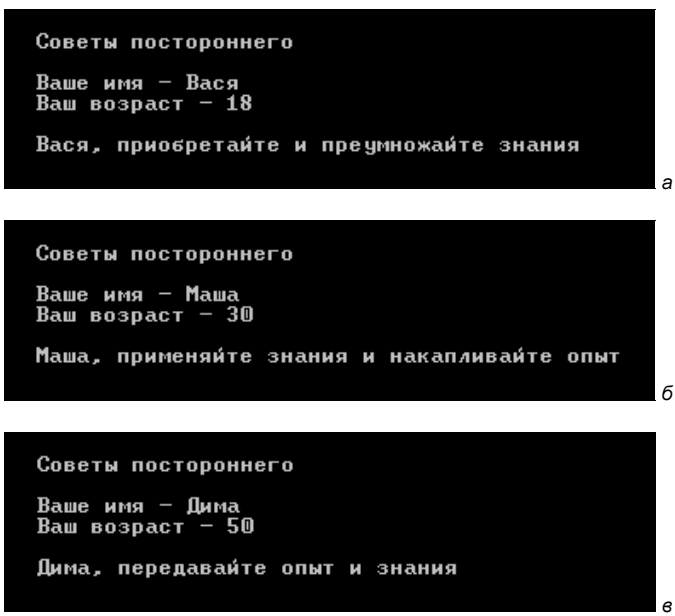


Рис. 3.4. "Советующая" программа: молодой возраст — до 23 лет (*а*); средний возраст — от 23 до 40 лет (*б*); зрелый возраст — после 40 лет (*в*)

Ввод данных осуществляется посредством операторов `INPUT` в строках 40 и 50, причем в строке 40 задается значение символьной переменной (имя), а в строке 50 — числовой (возраст). В зависимости от анализа с помощью оператора `SELECT CASE` введенного возраста, оператор `PRINT` в строке 90 выдает соответствующий возрасту совет. Отметим, что оператор `CHR$(44)` в операторе `PRINT` выводит запятую.

В примере 3.2 ввод значений символьных переменных производился посредством оператора присваивания. Посмотрим теперь, как подобный ввод можно осуществить с помощью оператора `INPUT`.

Пример 3.5. Ввод хокку

Требуется разработать программу для ввода хокку.

Программа 3.5

```
'3I-2opV.bas
10 DIM c$(3)
20 CLS
30 IF a = 0 THEN PRINT " Ввод хокку" ELSE PRINT "     Хокку"
40 PRINT
'===== Блок ввода и проверки ввода ================================
50 FOR i = 1 TO 3
60 IF a = 1 THEN  ELSE 90
70 PRINT c$(i)
80 GOTO 110
90 PRINT i; "- строка"
100 INPUT " ", c$(i)
110 NEXT i
120 PRINT
130 IF a = 0 THEN a = 1: GOTO 20
'====================================================================
140 END
```

Хокку — японское нерифмованное трехстишие, поэтому размер символьного массива в строке 10 задается равным 3. В приведенной программе цикл FOR...NEXT в строках 50—110 используется два раза. В первый раз для построчного ввода хокку оператором INPUT в строке 100. Затем следы ввода стираются, и хокку выводится уже в виде слитного трехстишия с помощью оператора PRINT в строке 70. Для переключения режимов ввода и вывода используется переменная a. Оператор в строке 130 изменяет после первого прохода значение переключателя a и осуществляет последующий возврат для контрольного вывода хокку, используя тот же цикл FOR...NEXT в строках 50—110.

Результат работы программы 3.5 дан на рис. 3.5.

а *б*

Рис. 3.5. Программа для ввода хокку: промежуточный результат после завершения построчного ввода (*а*); результат контрольного вывода хокку (*б*)

Если для ввода хокку уже пришлось использовать конструкцию с циклом, то для ввода числовых элементов матрицы требуется применение двойного (вложенного) цикла.

Пример 3.6. Ввод/вывод элементов матрицы

Требуется разработать программу для ввода элементов матрицы и проверочного вывода матрицы.

Программа 3.6

```
'3I-3opV.bas
10 CLS
20 PRINT "Задайте размеры матрицы"
30 INPUT " m = ", m
40 INPUT " n = ", n
50 DIM x(m, n)
'==1=== Блок ввода ================================================
60 PRINT "Ввод элементов матрицы"
70 FOR i = 1 TO m
80 FOR j = 1 TO n
90 PRINT SPC(3); "x ("; i; j; ")"
100 INPUT " = ", x (i, j)
110 NEXT j, i
'==2=== Блок проверки ввода ======================================
120 PRINT "   Матрица X"
130 FOR i = 1 TO m
140 PRINT SPC(3);
150 FOR j = 1 TO n
160 PRINT x (i, j);
170 NEXT j
180 PRINT
190 NEXT i
'=================================================================
200 END
```

С помощью операторов `INPUT` в строках 30—40 задаются размеры матрицы `x`. Затем посредством двойного цикла в строках 70—110, в тело которого помещен оператор `INPUT` (строка 100) с предшествующим ему оператором `PRINT` (строка 90), осуществляется ввод элементов матрицы `x`. Проверочный вывод матрицы производится также с помощью двойного цикла `FOR...NEXT`, в тело которого поставлен оператор `PRINT` (строка 160).

Результат работы программы 3.6 показан на рис. 3.6.

Рис. 3.6. Программа ввода/вывода элементов матрицы

Итак, ввод данных посредством оператора INPUT целесообразно производить в тех случаях, когда значения переменных могут меняться с каждым новым запуском программы.

3.3. Операторы *READ, DATA*

Если же данные не меняются с каждым новым запуском программы, но их объем велик для оператора присваивания, то рекомендуется ввод исходных осуществлять посредством операторов READ/DATA. Оператор DATA хранит числовые или символьные данные для их последующего чтения оператором READ. Оператор READ вводит переменные, для которых считывает и присваивает конкретные значения из оператора DATA. Тип данных в DATA должен соответствовать типу переменных в READ. Например:

```
10 READ x1, x2%, x3$, x4$
20 DATA 3.5, -6, "режим", 20
```

или

```
10 READ x1, x2, x3
20 DATA 1, , 2
```

В результате оператор READ в первом случае присвоит значения своим переменным в следующем порядке: x1 = 3.5, x2% = −6, x3$ = "режим", x4$ = 20. Во втором случае дан пример пустого элемента списка, и результат присваивания будет: x1 = 1, x2 = 0, x3 = 2.

Оператор READ допускается помещать в любом месте многооператорной строки, оператор DATA должен быть либо единственным, либо последним оператором строки. BASIC допускает чтение числовой константы в строковую переменную (как для x4$ в предыдущем примере).

Один оператор READ может читать несколько операторов DATA, и, наоборот, несколько операторов READ могут читать один оператор DATA. Если список переменных короче, чем список данных, то следующий оператор READ начинает читать первый непрочитанный элемент. Если данные не прочитаны никаким READ, то они игнорируются. Например:

```
10 READ x1, x2, x3, x4
20 DATA 1, 2, 3
30 DATA 4, 5
```

или

```
10 READ x1, x2
20 READ x3, x4
30 DATA 1, 2, 3, 4, 5
```

В приведенных примерах результат чтения и присваивания будет одинаковым: x1 = 1, x2 = 2, x3 = 3, x4 = 4, а 5 игнорируется.

Оператор RESTORE позволяет повторно считать данные из списка DATA. Например:

```
10 READ x1, x2, x3
20 RESTORE
30 READ x4, x5, x6
40 DATA 1, 2, 3
```

или

```
10 READ x1, x2, x3, x4, x5, x6
20 RESTORE 50
```

```
30 READ x7, x8
40 DATA 1, 2
50 DATA 3, 4
60 DATA 5, 6
```

В первом случае результат присваивания будет: x1 = 1, x2 = 2, x3 = 3, x4 = 1, x5 = 2, x6 = 3, поскольку RESTORE задает для READ из строки 30 возврат на считывание с начала данных из оператора DATA в строке 40. Во втором случае значения переменных будут следующими: x1 = 1, x2 = 2, x3 = 3, x4 = 4, x5 = 5, x6 = 6, x7 = 3, x8 = 4, поскольку метка 50 при RESTORE задает для READ из строке 30 переход на считывание с начала данных из оператора DATA в строке 50.

Сообщение "Ошибка в типе данных" вызывает попытка считать оператором READ из DATA строковую константу в числовую переменную. Числовое значение, слишком большое для переменной, приведет к ошибке "Переполнение" ("Overflow"). Если список переменных в операторах READ больше списка данных в операторах DATA, то это вызовет сообщение об ошибке "Нет данных" ("Out of DATA").

Использование операторов READ/DATA для ввода данных в программах рассмотрим на примерах.

Пример 3.7. Ввод числовой последовательности

Требуется разработать программу для ввода элементов числовой последовательности и ее проверочного вывода.

Программа 3.7

```
'3R-1opV.bas
10 CLS
20 PRINT "   Числовая последовательность"
30 DATA 3,-8,-4,3,7,5,-2,1,9,2
40 FOR j = 1 TO 10
50 READ c (j)
60 PRINT c (j);
70 NEXT j
80 END
```

Оператор READ в строке 50, поставленный в цикл в FOR...NEXT строках 40—70, считывает в переменную c (j) числа, содержащиеся в операторе DATA в строке 30. Знак точка с запятой, стоящий в операторе PRINT в строке 60, позволяет вывести числовую последовательность в одной строке.

Результат работы программы 3.7 приведен на рис. 3.7.

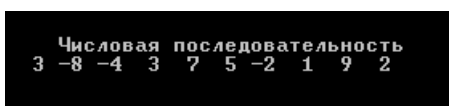


Рис. 3.7. Программа ввода и контрольного вывода числовой последовательности

Операторы READ/DATA производят автоматический ввод данных. Однако в программе можно также использовать одновременно и оператор INPUT, что позволяет сочетать автоматический

ввод с выбором. Такой подход хорош при работе с символьными переменными. Это соображение иллюстрирует следующий пример.

Пример 3.8. Анкета сотрудника

Требуется разработать программу "Анкета", в которой после выбора сотрудника автоматически выводятся его анкетные данные.

Программа 3.8

```
'3R-2opV.bas
10 CLS
20 PRINT "   Номера сотрудников"
30 PRINT "Петров — 1, Павлов — 2, Гор — 3"
40 PRINT "Шагин — 4, Зуева — 5"
50 INPUT "Номер сотрудника — ", n1
60 PRINT
70 PRINT "       Анкета"
80 PRINT
'==1===== Блок вывода анкеты ======================================
90 mx = 2
100 RESTORE 210
110 FOR i = 1 TO 2
120 FOR j = 1 TO 6
130 READ c$( j )
140 LOCATE 7 + j, mx
150 PRINT c$( j )
160 NEXT j
170 mx = 14
180 SELECT CASE n1
     CASE 1: RESTORE 220
     CASE 2: RESTORE 230
     CASE 3: RESTORE 240
     CASE 4: RESTORE 250
     CASE 5: RESTORE 260
190 END SELECT
200 NEXT i
'==2===== Блок данных ==============================================
210 DATA "Фамилия","Имя","Отчество"
       DATA "Должность","Год рожд.","Телефон"
220 DATA "Петров","Олег","Фомич","1974"
       DATA "артист","242-17-13"
230 DATA "Павлов","Иван","Силыч","1946"
       DATA "артист","564-54-89"
240 DATA "Гор","Алла","Юрьевна","1957"
       DATA "продюсер","342-12-90"
250 DATA "Шагин","Петр","Иванович","1965"
       DATA "режиссер","765-87-48"
260 DATA "Зуева","Зоя","Яновна","1967"
       DATA "актриса","654-74-92"
'====================================================================
270 END
```

Операторы PRINT в строках 20—40 выводят на экран своеобразное меню-подсказку перед выбором номера сотрудника с помощью оператора INPUT. Для обработки выбора и вывода анкеты образован двойной цикл FOR...NEXT. Внешний цикл в строках 110—200 со счетчиком i обеспечивает два прохода внутреннего цикла в строках 120—160 со счетчиком j. За первый проход оператором PRINT в строке 150 выводятся заголовки пунктов анкеты, которые берутся оператором READ в строке 130 из оператора DATA в строке 210, что определяет RESTORE в строке 100. Выбор номера сотрудника оператором INPUT в строке 50 обрабатывается оператором SELECT CASE в строках 180—190, в результате чего соответствующий оператор RESTORE определяет тот оператор DATA из строк 220—260, из которого и будут считываться анкетные данные выбранного сотрудника на втором шаге внутреннего цикла.

Результат работы программы 3.8 дан на рис. 3.8.

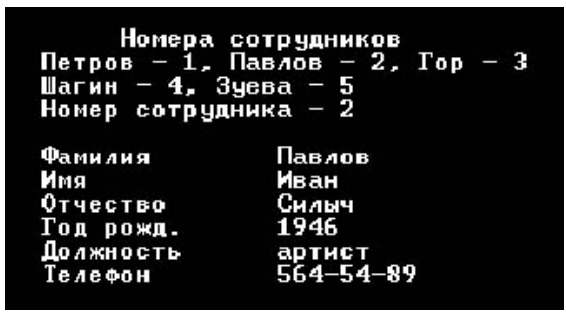


Рис. 3.8. Программа "Анкета"

Как видно из предыдущего примера, операторы READ/DATA позволяют применять для ввода данных такое мощное оружие, как циклы, что при использовании оператора присваивания весьма затруднительно. В программах для работы с графикой операторы READ/DATA применяются для ввода координат, например, точек.

Пример 3.9. Изображения созвездий

Требуется разработать программу, выводящую изображение созвездий Большая и Малая Медведицы.

Программа 3.9

```
'3R-3opV.bas
10 CLS        'очистка экрана
20 SCREEN 9   'установка экранного режима
30 DIM x (20), y (20)
'==1==== Блок рисования звезд ======================================
40 DATA -260,-130,  -10,4,  -12,8,  -8,12,  -14,0
50 DATA -4,12,  12,2,  100,50,  8,10,  -24,3
60 DATA -16,-12,  -16,0,  -20,-2,  -20,7
70 FOR i = 1 TO 14
80 READ x ( i ), y ( i )
90 LINE STEP(x ( i ), y ( i )) — STEP(2, 2), 14, BF
100 NEXT i
```

```
'==2==== Блок вывода названий ====================================
110 LOCATE 2, 3
120 PRINT "Малая Медведица"
130 LOCATE 14, 3
140 PRINT "Большая Медведица"
150 LOCATE 4, 10
160 PRINT "Полярная звезда"
'==================================================================
170 END
```

Поставленный в цикл FOR...NEXT в строках 70—100 оператор READ в строке 80 считывает значения координат звезд, рисуемых оператором LINE в строке 90, из оператора DATA в строках 40—60. Причем значения координат в DATA задаются как приращение к координатам последней точки рисования (поскольку LINE со STEP). Поэтому, меняя лишь значения первой пары (–260,–130) из оператора DATA в строке 40, можно переносить изображение всех четырнадцати звезд по экрану без изменения. Вывод названий звезды и созвездий позиционируется операторами LOCATE в строках 110, 130 и 150.

Результат работы программы 3.9 показан на рис. 3.9.

Рис. 3.9. Программа рисования созвездий

3.4. Другие возможности

Помимо рассмотренных ранее операторов существуют и другие операторы, которые также можно использовать для ввода данных. Первый такой оператор — LINE INPUT.

LINE INPUT — оператор, присваивающий вводимую строку (до 255 символов) символьной переменной без влияния знаков разделителей. Например:

```
LINE INPUT "Введите значение y "; y$
```

Символьная константа в кавычках — подсказка или приглашение на ввод данных. Можно включить в нее знак вопроса (?). Символьная переменная (в данном случае y$) принимает все символы до нажатия клавиши <Enter>.

В отличие от оператора INPUT, знак вопроса после приглашения к вводу данных не выводится. Знак точка с запятой перед приглашением сохраняет курсор на той же строке после нажатия клавиши <Enter>. Для редактирования вводимой строки оператора LINE INPUT используются те же клавиши, что и в операторе INPUT.

Пример 3.10. Различия в применении оператора *LINE INPUT*

Требуется разработать программу, иллюстрирующую отличия применения оператора LINE INPUT с точкой с запятой перед приглашением к вводу данных и без точки с запятой перед приглашением.

Программа 3.10

```
'3D-1opV.bas
10 CLS   'очистка экрана
20 PRINT "Применение точки с запятой"
30 PRINT "перед приглашением к вводу"
'==1===== С точкой с запятой ====================================
40 PRINT
50 PRINT "С точкой с запятой"
60 PRINT
70 LINE INPUT ; " 1-е слово — "; y1$
80 LINE INPUT ; " 2-е слово — "; y2$
'==2===== Без точки с запятой ===================================
90   PRINT
100 PRINT "Без точки с запятой"
110 PRINT
120 LINE INPUT " 1-е слово — "; y3$
130 LINE INPUT " 2-е слово — "; y4$
'================================================================
140 END
```

Программа 3.10 простая и прозрачная и особых комментариев не требует. Результат работы программы 3.10 приведен на рис. 3.10.

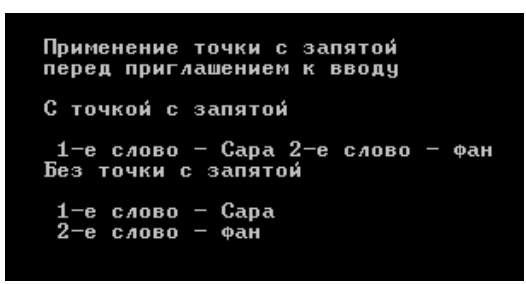


Рис. 3.10. Различия в применении оператора LINE INPUT

Следующая программа показывает, что оператор LINE INPUT позволяет вводить данные без влияния знаков разделителей.

Программа 3.11

```
'3D-2opV.bas
10 CLS   'очистка экрана
20 PRINT
30 PRINT "Ввод фраз со знаками"
40 PRINT "разделителями"
50 PRINT
60 LINE INPUT "1-я фраза — "; y1$
70 LINE INPUT "2-я фраза — "; y2$
80 PRINT
90 PRINT "  Проверка ввода"
100 PRINT
110 PRINT y1$
120 PRINT y2$
130 END
```

Результат работы программы 3.11 дан на рис. 3.11.

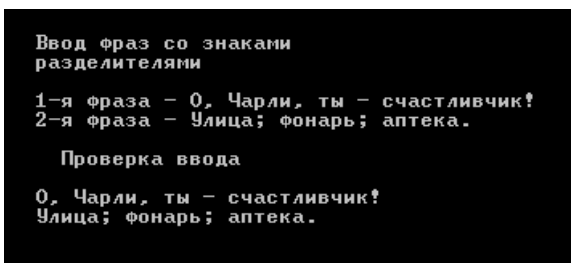


Рис. 3.11. Программа с операторами `LINE INPUT`

Другой оператор, который можно использовать для ввода данных, — это функция `INPUT$`.

`INPUT$` — функция ввода, читающая символьные строки с клавиатуры:

```
INPUT$(n)
```

где `n` — количество символов, читаемых с клавиатуры.

Знаки не дублируются на экран (ввод без "эха"). Данная функция ждет ввода указанного числа символов, после чего не требуется нажатия клавиши <Enter>.

Пример 3.11. Задержка до нажатия любой клавиши

В примере реализована задержка исполнения программы до нажатия клавиши.

Программа 3.12

```
'3D-3opV.bas
10 CLS   'очистка экрана
20 PRINT "Для продолжения нажмите любую клавишу"
30 c$ = INPUT$(1)
40 PRINT
50 PRINT "Работа программы продолжена"
60 END
```

Результат работы программы 3.12 дан на рис. 3.12.

Поскольку ввод с помощью функции INPUT$ не дублируется на экран, то для проверки ввода следует использовать оператор PRINT. Это реализовано в следующей программе.

Программа 3.13

```
'3D-4opV.bas
10 CLS   'очистка экрана
20 PRINT "  Введите 5 символов"
30 c$ = INPUT$(5)
40 PRINT
50 PRINT " Проверка ввода"
60 PRINT "     "; c$
70 END
```

Результат работы программы 3.13 приведен на рис. 3.13.

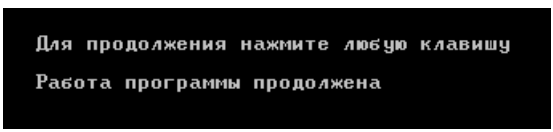


Рис. 3.12. Программа с задержкой до нажатия клавиши

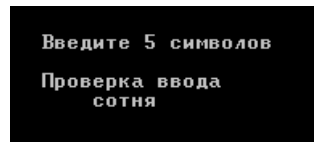


Рис. 3.13. Ввод 5 символов с проверкой ввода

INKEY$ — функция ввода, читающая символы с клавиатуры. Функция возвращает однобайтовую или двухбайтовую строку, содержащую символ, считанный с клавиатуры. Если символ не был считан, то возвращается нулевая строка. Функция INKEY$ не ожидает ввода в отличие от функции INPUT$. Функция INKEY$ не дублирует ввод с клавиатуры выводом на экран, т. е. ввод осуществляется без "эха". В табл. 3.2 представлены клавиши, нажатие которых обрабатывается особо.

Таблица 3.2. *Обработка нажатия клавиш*

Клавиши	Действие при обработке
<Ctrl>+<Break>	Останавливает выполнение
<Ctrl>+<Alt>+<Delete>	Перезагрузка компьютера
<Ctrl>+<NumLock>	Пауза
<Print Screen>	Печать содержимого экрана

Пример 3.12. Ввод пароля

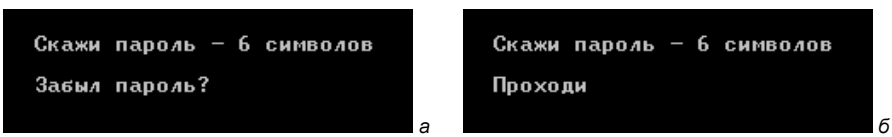
Требуется разработать программу для ввода пароля.

Программа 3.14

```
'3D-5opV.bas
10 DIM c$(6)
```

```
20 CLS   'очистка экрана
30 PRINT "Скажи пароль — 6 символов"
'==1===== Блок ввода пароля ======================================
40 FOR i = 1 TO 6
50 SLEEP
60 c$ (i) = INKEY$
70 SELECT CASE c$ (i)
     CASE "G", "g", "П", "п"
                IF i = 1 THEN a1 = 1
     CASE "F", "f", "A", "a"
                 IF i = 2 THEN a2 = 1
80 END SELECT
90 NEXT i
'==2===== Блок вывода сообщения ===================================
100 PRINT
110 IF a1 = 1 AND a2 = 1 THEN  ELSE 130
120 PRINT "Проходи": GOTO 140
130 PRINT "Забыл пароль?": SLEEP 3: GOTO 20
'================================================================
140 END
```

Результат работы программы 3.14 приведен на рис. 3.14.



а *б*

Рис. 3.14. Программа ввода пароля: пароль введен неправильно (*а*); пароль введен правильно (*б*)

Функция `INKEY$` часто используется для формирования условия выхода из цикла. Например, следующий цикл будет работать до нажатия любой клавиши:

```
DO
(тело цикла)
LOOP WHILE INKEY$ = ""
```

Этот цикл выполняется до нажатия клавиши <Esc>:

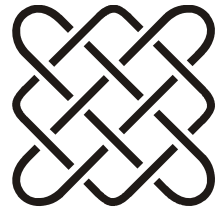
```
WHILE INKEY$ <> CHR$(27)
(тело цикла)
WEND
```

Также функция `INKEY$` обычно используется для обработки нажатия клавиш. Так, в программе игры, в которой фигура человечка перемещается по горизонтали в любую сторону, задействуются клавиши "влево" и "вправо". Покажем фрагмент программы, в котором реализуется обработка этих клавиш:

```
100 DO
110 A$ = INKEY$
120 LOOP WHILE A$ = ""
130 kod = ASC(RIGHT$(A$, 1))
140 IF kod = 75 AND kod = 77 THEN ELSE 100
```

```
150 IF kod = 75 THEN c$ = "влево"
160 IF kod = 77 THEN c$ = "вправо"
170 PRINT "Нажата клавиша — " + c$
```

В строках 100—120 организован цикл DO...LOOP, выполняющийся до нажатия любой клавиши (условие WHILE A$ = "" при LOOP в строке 120). Функция фиксирует нажатие клавиши и передает в переменную A$. Строка 140 — то, что в литературе по программированию обычно именуется "защитой от дурака", а по версии авторов — "защита от любознательного". Строки 150—160 предназначены для распознавания нажатой клавиши по коду.



ГЛАВА 4

Вывод данных

4.1. Оператор *PRINT*

Вывод результата на экран дисплея осуществляется в BASIC посредством оператора PRINT, используемого практически в любой программе. Этот оператор столь многообразен в своих применениях, что его следует рассмотреть подробнее.

PRINT — оператор ввода/вывода, выдающий данные на экран.

```
PRINT x1, x2, ... , xN
```

x1, x2, ... , xN — список выражений любого типа, кроме пользовательского. Символьные константы в списке выражений должны заключаться в кавычки. Если список выражений указан, то значения выражений выводятся на экран по порядку.

Самый простой пример:

```
10 PRINT 8
```

выведет на экран число 8.

Перед отрицательным числом PRINT ставит знак минус, а перед положительным — пробел. Элементов в списке оператора PRINT может быть несколько. Оператор PRINT без списка элементов выводит на экран строку пробелов (пустую строку). Это иногда необходимо, например, для отделения заголовка таблицы от данных.

Если после оператора PRINT записано арифметическое или алгебраическое выражение, то оно будет вычислено, и результат будет выведен на экран. Например:

```
10 PRINT 5 * 5
```

вычислит и выведет на экран число 25, а строка кода:

```
10 PRINT (10 – 5) / (1 + 1.5)
```

вычислит и выведет на экран 2.

Следующий код вычислит и выведет на экран число 11, а затем выведет на экран 10 и 5 (в списке элементов оператора PRINT в строке 20 их 3).

```
10 READ y: DATA 5
20 PRINT 2 * y + 1, 10, 5
```

BASIC печатает результаты вычислений в виде десятичного числа (целого или с плавающей запятой), если эти результаты находятся в интервале от 0.01 до 999 999. В остальных случа-

ях BASIC использует экспоненциальное представление. Примеры отображения чисел приведены в табл. 4.1.

Таблица 4.1. *Примеры вывода чисел*

Вводимое значение	Значение, печатаемое BASIC
0.000789	7.89000E-04
0.03	.03
999889	999889
2200000	2.20000E+06

Строка символов, выводимая на экран, состоит из 5 зон по 14 позиций в каждой зоне. Если элементы в списке оператора `PRINT` разделены запятыми, то каждый последующий элемент печатается в следующей свободной зоне. Когда последняя зона в строке заполнена, BASIC продолжает печать с 1-й зоны следующей строки. Наличие в списке оператора `PRINT` двух рядом стоящих запятых вызывает пропуск одной зоны.

Пример 4.1. Использование разных разделителей в операторе *PRINT*

Далее приведена программа, показывающая вид вывода на экран в зависимости от знаков, разделяющих элементы в списке оператора `PRINT`.

Программа 4.1

```
'4znak.bas
10 CLS
20 PRINT "Разделители — запятые"
30 PRINT "x = ", 5, "y = ", 3, "z = ", 2
40 PRINT
50 PRINT "Разделители — точки с запятой"
60 PRINT "x = "; 5; "y = "; 3; "z = "; 2
70 PRINT
80 PRINT "Разделители — комбинация знаков"
90 PRINT "x = "; 5, "y = "; 3, "z = "; 2
100 END
```

Результат работы программы 4.1 дан на рис. 4.1.

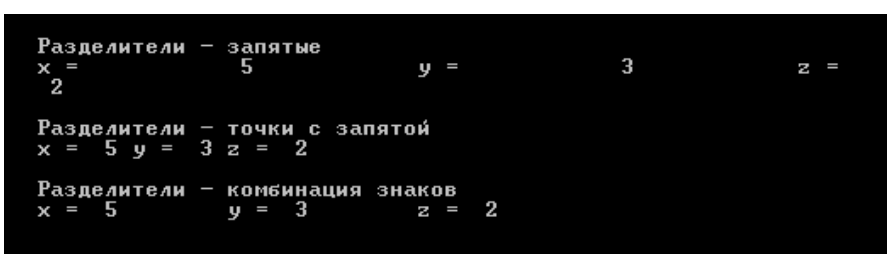


Рис. 4.1. Вид вывода в зависимости от вида разделителей

Разделение элементов списка оператора PRINT точкой с запятой позволяет выводить их в одной строке один рядом с другим (печать без перевода строки), а также печатать результат рядом с символьной строкой. Например, код:

```
10 READ y: DATA 5
20 PRINT "z = "; 2 * y + 1
```

выведет на экран: z = 11.

Заключенные в кавычки строковые выражения печатаются в том виде, в каком они представлены в операторе PRINT. Например, следующий код:

```
20 READ y: DATA 3
30 PRINT "Площадь квадрата S = "; y ^ 2; "кв. см"
```

выведет на экран: Площадь квадрата S = 9 кв. см. Поэтому необходимо не забывать о пробелах внутри кавычек для красивого вывода (в строке 30 предыдущего примера — это пробел между знаком равно (=) и закрывающими кавычками). Задавая пробелы внутри кавычек, можно управлять видом вывода:

```
10 PRINT "    Таблица"
```

выведет на экран слово "Таблица" со сдвигом на 7 пробелов.

В записанных в операторе PRINT выражениях возможно использование тригонометрических или иных функций, а также функций округления. Например:

```
10 READ y: DATA 3
20 PRINT CINT(2 * y ^ 2 / SIN (y)); 2 * y ^ 2 / SIN (y)
```

выведет на экран 128 (округленное) и 127.551. Функции, выводящие результат в виде целого числа, даны в табл. 4.2.

Таблица 4.2. *Операторы округления чисел*

Число n	**–1.99**	**–1.5**	**–1.01**	**1.01**	**1.5**	**1.99**	**Примечание**
CINT(n)	–2	–2	–1	1	2	2	Округление
FIX(n)	–1	–1	–1	1	1	1	Отсечение дробной части
INT(n)	–2	–2	–2	1	1	1	Ближайшее целое, меньшее или равное аргументу

Чтобы вывести таблицей элементы матрицы (двумерного массива), необходимо поставить оператор PRINT в двойной цикл, например:

```
40 FOR i = 1 TO 4
50 FOR j = 1 TO 3
60 PRINT m(i, j);
70 NEXT j
80 PRINT
90 NEXT i
```

Точка с запятой после m(i, j) в строке 60 позволяет вывести на экран в одной строке все элементы m(1, j). После чего PRINT из строки 80 обрывает экранную строку, и 60 PRINT m(i, j); выводит уже на следующей экранной строке все элементы m(2, j), 80 PRINT обрывает 2-ю экранную строку и т. д.

4.2. Операторы, совместимые с оператором *PRINT*

Вид вывода оператором PRINT может регулироваться с помощью функции TAB.

TAB — функция ввода/вывода, сдвигающая позицию вывода при использовании операторов PRINT и LPRINT:

```
TAB (n)
```

где n — числовое выражение в пределах от 1 до ширины вывода, новая колонка ввода в данной строке.

Если указана позиция, находящаяся левее текущей, то BASIC игнорирует требование функции TAB. Если n больше, чем ширина вывода, то вывод начнется с позиции, определяемой выражением:

```
1 + (n MOD h)
```

где h — ширина вывода.

Если n < 1, то вывод начнется с позиции 1.

Следующий пример выведет 2 в 7-й позиции, 3 в 18-й позиции и 4 — в 23-й:

```
10 x = 2: y = 3: z = 4
20 PRINT TAB (7); x; TAB (18); y; TAB (23); z
```

Вариант вывода матрицы с использованием функции TAB приведен в следующей программе.

Пример 4.2. Вывод элементов матрицы с помощью табуляции

Рассмотрим программу, выводящую элементы матрицы таблицей.

Программа 4.2

```
'4-2viv.bas
10 CLS
20 DATA 1,2,3,4,5,6,7,8,9
30 PRINT "  Матрица A"
40 FOR i = 1 TO 3
50 FOR j = 1 TO 3
60 READ m(i, j)
70 PRINT TAB(3 * j); m(i, j);
80 NEXT j
90 PRINT
100 NEXT i
110 END
```

Результат работы программы 4.2 представлен на рис. 4.2.

Рис. 4.2. Вывод элементов матрицы с помощью табуляции

Ввод элементов матрицы осуществляется посредством пары операторов READ/DATA в строках 60 и 20 соответственно, а вывод — оператором PRINT. Расстояние между столбцами определяет функция TAB в строке 70.

Для регулирования вывода также существует SPC(n) — функция ввода/вывода, пропускающая пробелы в текущей строке при выполнении оператора PRINT:

```
SPC(n)
```

где n — числовое выражение в пределах от 0 до 32 767, обозначающее количество пробелов, пропускаемых в текущей строке.

Следующий пример выведет на экран: Очень Очень (с пятью пробелами между словами):

```
PRINT "Очень"; SPC(5); "Очень"
```

Если функции TAB и SPC регулируют вывод "внутри" оператора PRINT, то позицию начала вывода на экране можно задать с помощью оператора LOCATE.

LOCATE — оператор ввода/вывода, передвигающий курсор в указанную позицию экрана:

```
LOCATE y, x
```

где x — номер столбца экрана ($0 \le x \le 80$), если параметр не указан, то номер столбца не меняется; y — номер строки ($0 \le y \le 25$), если параметр не указан, то номер строки не меняется.

Можно опустить любой параметр в операторе. Если опустить строку и столбец, то оператор LOCATE оставит курсор на позиции, установленной предыдущим оператором LOCATE или предыдущим оператором ввода/вывода. Для других аргументов сохраняются предыдущие значения.

Например, для вывода в центре экрана цифры 0 следует написать:

```
10 LOCATE 25, 40
20 PRINT 0
```

Рассмотрим следующую небольшую программу с использованием оператора LOCATE.

Пример 4.3. Вывод данных в виде таблицы

Следующая программа выводит данные таблицей.

Программа 4.3

```
'4-1viv.bas
10 CLS
20 LOCATE 1, 11: PRINT "Таблица"
30 FOR y = 3 TO 5
40 FOR x = 10 TO 16 STEP 2
50 LOCATE y, x: PRINT y
60 NEXT x, y
70 END
```

Результат работы программы 4.3 представлен на рис. 4.3.

Позицию вывода определяют операторы LOCATE в строках 20 и 50. Параметры в операторе LOCATE в строке 50 задают циклы FOR...NEXT в строках 30—60. В данном примере осуществляется регулярный вывод.

Рис. 4.3. Вывод данных в виде таблицы

Параметрами в операторе могут быть числа или числовые выражения, переменные, как в только что рассмотренном примере, а также элементы массива, что дает возможность использовать циклы для нерегулярного вывода, когда для значений параметров нет математических закономерностей. Поясним это на примере.

Пример 4.4. Нерегулярный вывод

Программа выполняет нерегулярный вывод на экран.

Программа 4.4

```
'9LOCATE1.bas
10 CLS
20 DIM x(5)
30 DIM y(5)
40 DATA 3,5,3,27,4,19,6,7,7,23
50 FOR i = 1 TO 5
60 READ y(i), x(i)
70 LOCATE y(i), x(i)
80 PRINT "хорошо"
90 NEXT i
100 END
```

Результат работы программы 4.4 показан на рис. 4.4.

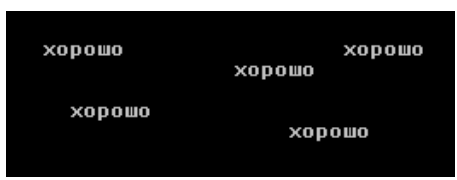


Рис. 4.4. Пример нерегулярного вывода

Порядок значений `y(i)`, `x(i)`, задаваемый оператором `DATA` в строке 40, фиксируется в массивах и обрабатывается с помощью оператора `LOCATE` в строке 70, входящего в тело цикла `FOR...NEXT` в строках 50—90. Оператор `LOCATE` в строке задает позицию вывода для оператора `PRINT` в строке 80.

4.3. Оператор *PRINT USING*

В предыдущем разделе были даны примеры вывода двумерных массивов в виде таблицы. Если же элементами массива могут быть как однозначные, так и двузначные числа, то с этой целью лучше использовать `PRINT USING` — оператор отформатированного вывода.

`PRINT USING` — оператор ввода/вывода, осуществляющий вывод на экран по указанному формату:

```
PRINT USING f$; x1, x2, ... , xN
```

где `f$` — формат — символьное выражение, содержащее специальные символы, управляющие представлением выводимых данных;

`x1, x2, ... , xN` — список выражений любого типа, кроме пользовательского. Символьные константы в списке выражений должны заключаться в кавычки.

В табл. 4.3 даны варианты использования этого оператора.

Таблица 4.3. *Оператор* `PRINT USING` *и его описание*

Оператор	Вывод на экран
`PRINT USING "Число ###"; 16`	`Число  16` где `#` — позиция цифры
`PRINT USING "Число #"; 16`	`Число # (переполнение)` или `%16`
`PRINT USING "Число +##"; 16`	`Число +16` где `+` печатает знак минус для отрицательных чисел и знак плюс для положительных
`PRINT USING "Число $$####"; 16`	`Число $16` где `$$` печатает `$` перед выводимым числом
`PRINT USING "Число **####"; 16`	`Число ****16` где `**` заполняет звездочками места перед числом
`PRINT USING "Число **$####"; 16`	`Число ****$16` где `**$` совмещает `**` и `$$`
`PRINT USING "Ответ: #.### кг."; x`	`Ответ: 3.041 кг.` (для x = 3.0406) и `Ответ: %40.000 кг.` (для x = 40) где `.` (точка) — позиция десятичной точки
`PRINT USING "Ответ: #.##^^^^ кг."; x`	`Ответ: .30E+01 кг.` (для x = 3.0406) и `Ответ: .40E+02 кг.` (для x = 40) где `^^^^` — позиция для печати степени (числа в экспоненциальном представлении или в так называемом научном формате)
`PRINT USING "!, &, \ \ , _&"; "Москва", "Москва", "Москва"`	`М, Москва, Мос, &` где `!` — печать только первого символа, `&` — место для неограниченного строкового выражения, `\ \` — печать первых `n` символов строкового выражения (`n` — число пробелов между косыми чертами + 2), `_` — следующий символ воспринимается не как управляющий, а как простой литерал

Примечание

`#, !, $` и др. — управляющие символы. Остальные символы в управляющей строке печатаются без изменений.

Простой пример:

```
20 y = 3
30 PRINT USING "Площадь квадрата S = # кв.см"; y ^ 2
```

Оператор PRINT USING вычислит и выведет на экран:

```
Площадь квадрата S = 9 кв.см
```

Можно использовать оператор PRINT USING и для организации красивого ввода данных.

Пример 4.5. Ввод данных с помощью оператора *PRINT USING*

Следующая программа осуществляет ввод данных.

Программа 4.5

```
'9PRINTU1.bas
10 CLS
20 PRINT "  Введите значения x"
30 FOR i = 2 TO 17 STEP 3
40 IF i < 10 THEN PRINT USING "  x#"; i;
50 IF i > 9 THEN PRINT USING "  x##"; i;
60 INPUT " = ", x
70 NEXT i
80 END
```

Результат работы программы 4.5 дан на рис. 4.5.

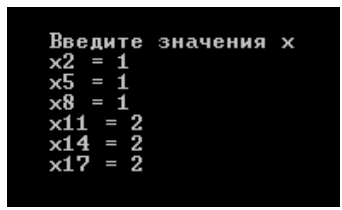


Рис. 4.5. Пример ввода данных

Покажем, как можно использовать оператор PRINT USING для вывода матрицы таблицей.

Пример 4.6. Вывод матрицы в виде таблицы

Следующая программа осуществляет вывод матрицы таблицей.

Программа 4.6

```
'9PRINTU2.bas
10 CLS
20 PRINT "   Матрица A"
30 DATA 12,3,6,15,22,6,4,11,14
40 FOR j = 1 TO 3
50 FOR i = 1 TO 3
60 READ x(j, i)
70 PRINT USING "####"; x(j, i);
80 NEXT i
90 PRINT
100 NEXT j
110 END
```

Результат работы программы 4.6 показан на рис. 4.6.

Расстояние между столбцами определяется количеством решеток в операторе PRINT USING в строке 70. В остальном программа сходна с предыдущими аналогичными программами.

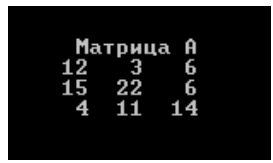


Рис. 4.6. Вывод матрицы таблицей с помощью оператора PRINT USING

Предлагаем читателю самостоятельно опробовать остальные возможности по форматированному выводу данных, предоставляемые оператором PRINT USING.

4.4. Другие возможности

WRITE — оператор ввода/вывода, посылающий данные на экран:

```
WRITE x1, x2, ... , xN
```

где x1, x2, ... , xN — список выражений, содержащий одно или несколько выражений, разделенных запятыми.

Если список выражений опущен, то печатается пустая строка. Выводимые значения разделяются запятыми, символьные строки заключаются в кавычки. После печати последнего выражения в списке вставляются символы возврата каретки и перевода строки. Оператор WRITE записывает числовые значения без начальных и конечных пробелов.

Пример 4.7. Различия между *WRITE* и *PRINT*

Рассмотрим программу, иллюстрирующую различия между операторами WRITE и PRINT.

Программа 4.7

```
'4-2viv.bas
10 CLS
20 PRINT "  Различие между WRITE и PRINT"
30 PRINT
40 a = 20
50 b = 50
60 c$ = "Информатика"
70 d = -1.2E-11
80 WRITE a, b, c$, d
90 PRINT a, b, c$, d
100 END
```

Результат работы программы 4.7 приведен на рис. 4.7.

CSRLIN — функция ввода/вывода, возвращающая значение текущей строки, в которой находится курсор:

```
n = CSRLIN
```

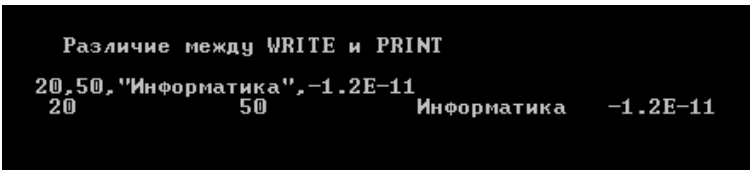


Рис. 4.7. Программа, иллюстрирующая различия между `WRITE` и `PRINT`

`POS` — функция ввода/вывода, возвращающая номер столбца, в котором находится курсор:

```
POS(0)
```

Пример 4.8. Использование функций *POS* и *CSRLIN*

Следующая программа обеспечивает набор четырех строк по 35 знаков в каждой.

Программа 4.8

```
'4-4VIV.bas
10 CLS
20 PRINT : PRINT "  Нажмите Esc для завершения"
30 PRINT
'==1===== Цикл набор текста ======================================
40 DO
50 PRINT SPC(5);
    '--2----- Цикл, обеспечивающий введение строки ----------------
60 DO WHILE POS(0) < 41
    '--3----- Цикл фиксации нажатия клавиши ------------------------
70 DO
80 k$ = INKEY$
90 LOOP WHILE k$ = ""
    '-----------------------------------------------------3-------
100 IF ASC(k$) = 27 OR CSRLIN > 7 THEN 140 ELSE PRINT k$;
110 LOOP
    '-----------------------------------------------------2-------
120 PRINT
130 LOOP
'=========================================================1=======
140 END
```

Результат работы программы 4.8 показан на рис. 4.8.

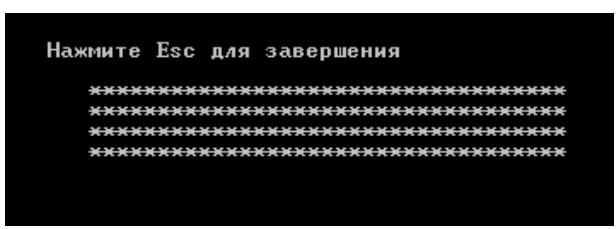


Рис. 4.8. Пример использования функций `POS` и `CSRLIN`

Программа 4.8 представляет собой три цикла DO...LOOP, два из которых вложенные. Вложенный цикл в строках 70—90 содержит в своем теле функцию INKEY$, обеспечивающую фиксацию нажатия клавиши. Если не нажата клавиша <Esc> и позиция набора не превысила седьмой строки, то выводится соответствующий символ. Выполнение условия окончания работы программы контролирует условный оператор IF...THEN в строке 100, используя функции ASC и CSRLIN. Нужное количество знаков в строке обеспечивает условие POS(0) < 41 в строке 60 и задаваемый в строке 50 отступ на пять знаков. На рис. 4.8 все допустимое поле заполнено символом '*' (звездочка), между тем как для его заполнения может быть использован любой другой символ по желанию разработчика.

До сих пор рассматривались операторы, обеспечивающие вывод данных на экран компьютера. Однако в языке BASIC возможен и вывод данных на принтер.

LPOS — функция ввода/вывода, возвращающая текущую позицию печатающей головки принтера в буфере принтера:

```
LPOS(n)
```

Здесь n — числовое выражение от 0 до 3, указатель порта принтера. 0 или 1 означает LPT1:, 2 означает LPT2:, и т. д.

Текущая позиция, возвращаемая функцией LPOS, может не совпадать с физическим положением печатающей головки, т. к. принтеры могут не воспринимать знаков табуляции и иметь буфер для печати.

LPRINT — оператор ввода/вывода, выдающий данные на принтер:

```
LPRINT x1, x2, ... , xN
```

где x1, x2, ... , xN — список выражений, разделенных точкой с запятой или запятой. Символьные константы в списке выражений должны заключаться в кавычки. Если список выражений указан, то значения выражений выводятся на принтер по порядку.

LPRINT USING — оператор ввода/вывода, осуществляющий вывод на принтер по указанному формату:

```
LPRINT USING f$; x1, x2, ... , xN
```

где x1, x2, ... , xN — список выражений, разделенных точкой с запятой или запятой; f$ — формат — символьное выражение, содержащее специальные символы, управляющие представлением выводимых данных.

Операторы LPRINT и LPRINT USING работают аналогично операторам PRINT и PRINT USING, но вывод данных идет на принтер. Если все аргументы опущены, то на принтер выводится пустая строка. Оператор LPRINT по умолчанию подразумевает принтер с шириной строки 80 знаков. Ширину строки можно изменить оператором WIDTH LPRINT.

WIDTH — оператор ввода/вывода, устанавливающий ширину строки вывода файла или устройства, а также количество строк и столбцов экрана.

Простой пример установки ширины печати в 132 символа:

```
WIDTH LPRINT 132
```

Параметры оператора WIDTH приведены в табл. 4.4.

В табл. 4.5 приведены примеры использования оператора WIDTH.

Таблица 4.4. *Оператор* `WIDTH`

Синтаксис	Описание
`WIDTH [столбцы],[строки]`	Устанавливает количество столбцов и строк экрана. Число столбцов может быть 40 или 80. По умолчанию 80. Число строк может быть 25, 30, 43, 50 или 60, в зависимости от адаптера и режима экрана. По умолчанию 25
`WIDTH #n, m`	`n` — номер файла, открытого оператором `OPEN`; `m` — ширина вывода. Устанавливает ширину вывода устройства, открытого как файл (например, LPT1: или CONS:)
`WIDTH u$, n`	`u$` — символьная константа, определяющая устройство (например, "CONS:"). `n` — ширина вывода. Устанавливает ширину вывода устройства, заданного символьной константой. Это определение действует до открытия устройства как файл оператором `OPEN`. Не действует, если устройство уже открыто
`WIDTH LPRINT n`	`n` — ширина вывода. Устанавливает ширину вывода принтера для всех последующих операторов `LPRINT`

Таблица 4.5. *Примеры использования оператора* `WIDTH`

Оператор	Комментарий
`WIDTH 40, 25`	40 столбцов, 25 строк
`WIDTH 80, 43`	80 столбцов, 43 строки
`WIDTH 80, 50`	80 столбцов, 50 строк
`SCREEN 12: WIDTH 80, 60`	80 столбцов, 60 строк
`WIDTH 80, 25`	Возврат к исходному состоянию 80×25

ГЛАВА 5

Условные операторы

В жизни человеку часто приходится сталкиваться с ситуациями, когда в зависимости от складывающейся обстановки ему нужно выбрать тот или иной способ действия. При разработке программы также довольно часто случаются ситуации, в которых программа должна выполнять какие-то действия в зависимости от определенных условий. В этом случае в программе используется структура, называемая *ветвлением*. В отличие от программ линейной структуры, в которых все операторы выполняются последовательно друг за другом, в программах с ветвлением в зависимости от значений исходных данных может работать только одна из двух или нескольких ветвей (фрагментов программы). Такая возможность реализуется в программе посредством условных операторов. В BASIC такими операторами являются `IF...THEN` и `SELECT CASE`.

5.1. Оператор *IF...THEN*

`IF...THEN...ELSE` (ЕСЛИ...ТОГДА...ИНАЧЕ) — управляющий оператор, осуществляющий условное ветвление операций, основанное на оценке логического выражения, которое может быть истинным или ложным. Оператор `IF...THEN` различается на блочную форму и линейную, а последняя — на полную и неполную (рис. 5.1).

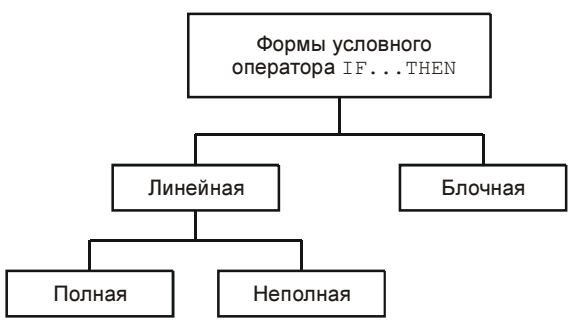


Рис. 5.1. Классификация форм оператора `IF...THEN`

Полная линейная форма в общем виде выглядит следующим образом:

```
IF (условие) THEN (операторы 1) ELSE (операторы 2)
```

Логика работы этого условного оператора следующая: если выполняется условие (*условие* истинно), то выполняются операторы, стоящие после THEN (*операторы 1*). Если же условие не выполняется (*условие* ложно), то выполняются операторы, стоящие после ELSE (*операторы 2*). После выполнения *операторов 1* или *операторов 2* программа переходит на следующую после оператора IF...THEN строку. Полная линейная форма особенно удобна при проверке альтернативного условия, такого как "Быть или не быть".

При формировании условий оператора IF...THEN используются операторы отношения, представленные в табл. 5.1.

Таблица 5.1. *Операторы отношения*

Оператор	Значение	Вид записи в BASIC
=	Равенство	X = Y
<>	Неравенство	X <> Y
<	Меньше	X < Y
>	Больше	X > Y
<=	Меньше или равно	X <= Y
>=	Больше или равно	X >= Y

В операторах IF...THEN могут быть и сложные условия, задаваемые с помощью логических операторов, приведенных в табл. 5.2.

Таблица 5.2. *Логические операторы BASIC*

Оператор	Название	Объяснение
NOT	Отрицание	NOT A истинно тогда и только тогда, когда A ложно
AND	Логическое умножение	A AND B истинно тогда и только тогда, когда истинно A и истинно B
OR	Логическое сложение	A OR B истинно тогда и только тогда, когда хотя бы одно из A и B истинно
XOR	Исключающее ИЛИ	A XOR B истинно тогда и только тогда, когда значения A и B не совпадают
EQV	Эквивалентность	A EQV B истинно тогда и только тогда, когда A и B одновременно истинны или ложны
IMP	Импликация	A IMP B ложно, если A истинно, а B ложно, в других случаях A IMP B истинно

Следует заметить, что не все интерпретаторы правильно реагируют на двойное условие. Поэтому их лучше записывать, используя логические операторы. Например, двойное условие $0 \le x \le 5$ можно записать следующим образом: 0 =< x AND x =< 5 или x < 0 EQV 5 < x.

Рассмотрим использование условных операторов в программах на примерах.

Пример 5.1. Полная линейная форма оператора *IF...THEN*

В следующей программе выполняется проверка времени прихода.

Программа 5.1

```
'5I-1uOp.bas
10 CLS   'очистка экрана
20 PRINT "Назовите время Вашего прихода"
30 INPUT "Сколько часов — ", Nch
40 INPUT "Сколько минут — ", Nm
50 CLS
60 PRINT "Время Вашего прихода "; Nch; "часов"; Nm; "минут"
70 v = Nch + Nm / 60
80 IF v <= 10.5 THEN c$ = "пришли вовремя" ELSE c$ = "опоздали"
90 PRINT "Вы "; c$
100 END
```

Программа 5.1 очень простая и прозрачная, поэтому практически не требует комментариев. Ввод времени прихода, производимый с помощью операторов в строках 20—40, стирается и осуществляется операторами `PRINT` в строках 60 и 90 упорядоченный вывод времени прихода и соответствующий комментарий, выбор которого обеспечивает оператор в строке 80.

Результат работы программы 5.1 представлен на рис. 5.2.

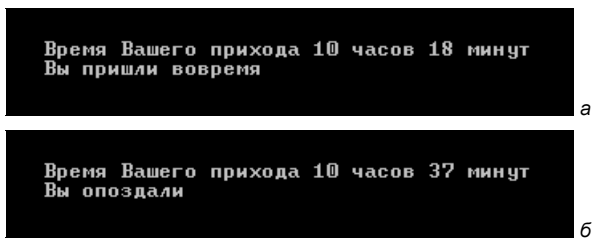


Рис. 5.2. Проверка времени прихода (ко 2-й паре):
время меньше контрольного (*а*); время больше контрольного (*б*)

При формировании условия оператора `IF...THEN` в строке 80 программы 5.1 используется числовая переменная. Однако с этой целью можно применять и символьные переменные, как показано в программе 5.2.

Программа 5.2

```
'5I-2uOp.bas
10 CLS   'очистка экрана
20 PRINT "Ленинград награжден орденом Дружбы народов?"
30 PRINT "да/нет — ";
40 o$ = INPUT$(1)
50 IF o$ = "L" OR o$ = "l" OR o$ = "Д" OR o$ = "д" THEN  ELSE 90
60 PRINT "да"
70 e = 1
80 GOTO 100
90 PRINT "нет"
100 PRINT
110 IF e = 1 THEN PRINT "Неправильно" ELSE PRINT "Правильно"
120 END
```

В программе 5.2 два оператора IF...THEN — в строках 50 и 110. Оператор в строке 50 анализирует результат нажатия клавиши, фиксируемый оператором INPUT$(1) в строке 40, ожидающим ввода с клавиатуры одного символа. Условие оператора IF...THEN в строке 50 сформировано из символьной переменной o$, логического оператора OR и оператора отношения (=). Оно фиксирует нажатие клавиши с русской буквой "д" независимо от регистра и раскладки клавиатуры. В принципе, строки 50—90 заменяются одним оператором IF...THEN, уходящим вправо за пределы экрана:

```
50 IF o$ = "L" OR o$ = "l" OR o$ = "Д" OR o$ = "д" THEN PRINT "да": e = 1
     ELSE PRINT "нет"
```

В операторе IF...THEN в строке 110 в качестве условия выступает значение переключателя e, а с помощью оператора PRINT выводится соответствующий комментарий.

Результат работы программы 5.2 дан на рис. 5.3.

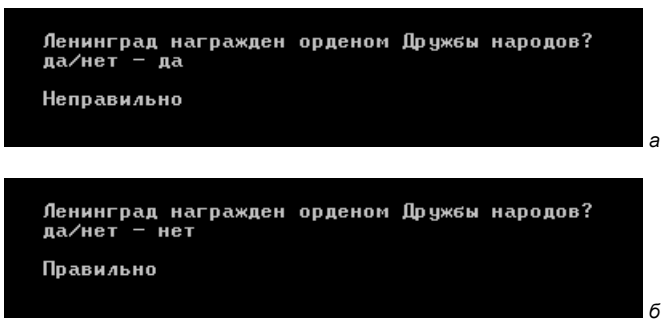


Рис. 5.3. Мини-тест: неправильный ответ (*а*); правильный ответ (*б*)

Неполная линейная форма в общем виде выглядит следующим образом:

```
IF (условие) THEN (операторы)
```

Логика работы этого условного оператора следующая: если выполняется условие (*условие* истинно), то выполняются операторы, стоящие после THEN (*операторы*). Если же условие не выполняется (*условие* ложно), то выполняется оператор, стоящий в программе на следующей строке после данного оператора IF...THEN. Например:

```
30 INPUT "x = ", x
40 IF x < 0 THEN y = x ^ 2: GOTO 60
50 y = 2 * x
60 PRINT "x = "; x
```

В приведенном фрагменте программы в зависимости от введенного в строке 30 значения x вычисление значения y происходит либо по формуле в строке 40 (при $x < 0$), либо по формуле в строке 50 (при $0 \le x$).

Если же в рассматриваемом фрагменте поменять местами строки 40 и 50, то результат его работы будет тот же, что обеспечивает оператор GOTO в строке 40 первого фрагмента.

```
30 INPUT "x = ", x
40 y = 2 * x
50 IF x < 0 THEN y = x ^ 2
60 PRINT "x = "; x
```

Нетрудно заметить, что результат работы операторов в строках 40—50 будет таким же, как и оператора полной линейной формы:

```
IF x < 0 THEN y = x ^ 2 ELSE y = 2 * x
```

Логика условий операторов `IF...THEN` наглядно проявляется при рассмотрении с помощью числовой оси. Рассмотрим следующий фрагмент программы:

```
30 INPUT "x = ", x
40 y = 2 * x
50 IF x < 0 THEN y = x ^ 2
60 IF x > 5 THEN y = x + 1
70 PRINT "x = "; x
```

Оператор в строке 40 не делит числовую ось и `y` вычисляется по этой формуле при любом значении `x` (рис. 5.4, *а*). Однако оператор `IF...THEN` в строке 50 уже делит числовую ось и при $x < 0$ значение `y`, посчитанное по формуле `y = 2 * x`, пересчитывается по формуле `y = x ^ 2` (рис. 5.4, *б*).

Оператор `IF...THEN` в строке 60 делит положительный отрезок числовой оси. При $5 < x$ значение `y`, посчитанное по формуле `y = 2 * x`, пересчитывается уже по формуле `y = x + 1` (рис. 5.4, *в*).

Таким образом, посчитанное по формуле `y = 2 * x` значение `y` не пересчитывается при $0 \le x \le 5$. При $x < 0$ значение `y` считается по формуле `y = x ^ 2`, а при $5 < x$ считается по формуле `y = x + 1`.

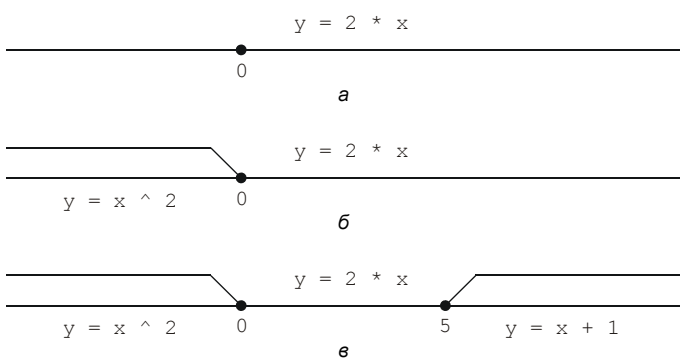


Рис. 5.4. Деление числовой оси на интервалы в соответствии с условием оператора `IF...THEN`: при выполнении строки 40 кода (*а*); строки 50 (*б*) и строки 60 (*в*)

Рассмотрим аналогичным образом следующий фрагмент:

```
30 INPUT "x = ", x
40 IF x < 0 THEN y = x ^ 2 ELSE y = 2 * x
50 IF x > 5 THEN y = x + 1
60 PRINT "x = "; x
```

Оператор `IF...THEN` в строке 40 делит числовую ось на две части (рис. 5.5, *а*).

При $0 < x$ значение `y` считается по формуле `y = x ^ 2`, а при $0 \le x$ считается по формуле `y = 2 * x`. Но оператор `IF...THEN` в строке 50 при $x > 5$ пересчитывает по формуле `y = x + 1` значение `y`, посчитанное ранее в строке 40 по формуле `y = 2 * x` (рис. 5.5, *б*). То есть осуществляемое ветвление аналогично ранее рассмотренному.

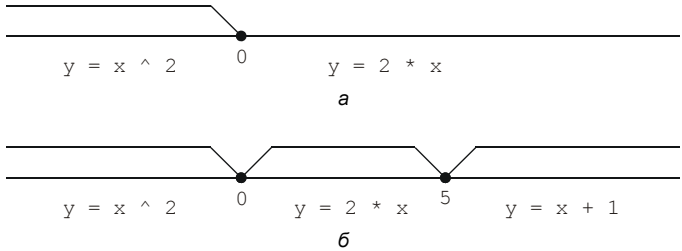


Рис. 5.5. Деление числовой оси на интервалы в соответствии с условием оператора `IF...THEN`: при выполнении строки 40 кода (*а*); строки 50 (*б*)

Рассуждая подобным образом, нетрудно убедиться, что подобное ветвление осуществит и следующий фрагмент:

```
30 INPUT "x = ", x
40 IF x >= 0 AND x <= 5 THEN y = 2 * x
50 IF x < 0 THEN y = x ^ 2
60 IF x > 5 THEN y = x + 1
70 PRINT "x = "; x
```

Следует заметить, что в операторе `IF...THEN` после `THEN` может следовать и другой `IF...THEN`. Тогда двойное условие $0 < x < 5$ может быть обеспечено следующим образом:

```
IF x > 0 THEN IF x < 5 THEN y = 2 * x
```

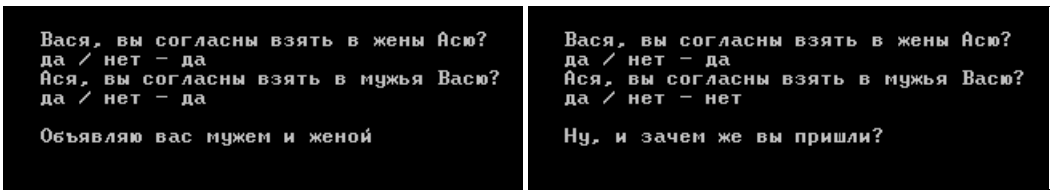
В программе иногда приходится использовать как полную линейную форму условного оператора, так и неполную. Пример использования неполной линейной формы условного оператора (строки 50 и 100) дан в программе 5.3.

Программа 5.3

```
'5I-3uOp.bas
10 CLS   'очистка экрана
'==1===== Блок решения Васи ========================================
20 PRINT "Вася, вы согласны взять в жены Асю?"
30 PRINT "да / нет — ";
40 v$ = INPUT$(1)
50 IF v$ = "L" OR v$ = "l" OR v$ = "Д" OR v$ = "д" THEN o1 = 1
60 IF o1 = 1 THEN PRINT "да" ELSE PRINT "нет"
'==2===== Блок решения Аси =========================================
70 PRINT "Ася, вы согласны взять в мужья Васю?"
80 PRINT "да / нет — ";
90 a$ = INPUT$(1)
100 IF a$ = "L" OR a$ = "l" OR a$ = "Д" OR a$ = "д" THEN o2 = 1
110 IF o2 = 1 THEN PRINT "да" ELSE PRINT "нет"
'==3===== Блок итога ===============================================
120 PRINT
130 IF o1 = 1 AND o2 = 1 THEN  ELSE 150
140 PRINT "Объявляю вас мужем и женой": GOTO 160
150 PRINT "Ну, и зачем же вы пришли?"
'====================================================================
160 END
```

Оператор INPUT$ в строке 40, ожидающий ввода с клавиатуры одного символа, фиксирует нажатие клавиши, а результат нажатия обрабатывается операторами IF...THEN в строках 50—60. Если нажата клавиша с русской буквой "д", то оператор в строке 60 выдаст ответ "да", при нажатии любой другой клавиши выдается ответ "нет". Точка с запятой после выражения в кавычках оператора PRINT в строке 30 обеспечивает вывод ответа в той же строке, что и выражение в кавычках "да / нет – ". Второй блок программы 5.3 аналогичен первому. Варианты ответов обрабатываются оператором IF...THEN в строке 130. Если оба ответа "да", то работает оператор PRINT в строке 140. Если хотя бы один ответ не "да", то резюме выводит оператор PRINT в строке 150.

Результат работы программы 5.3 приведен на рис. 5.6.



а *б*

Рис. 5.6. Опрос жениха и невесты: согласны оба (*а*); кто-то против (*б*)

Блочная форма оператора IF...THEN в общем виде выглядит следующим образом:

```
IF (условие 1) THEN
  (операторы 1)
ELSEIF (условие 2) THEN
  (операторы 2)
ELSEIF (условие 3) THEN
  (операторы 3)
 ...
ELSE (условие N) THEN
  (операторы N)
END IF
```

Логика работы условного оператора IF...THEN в блочной форме такая: сначала анализируется условие после IF (*условие 1*). Если оно выполняется (*условие 1* истинно), то выполняются операторы блока THEN. Если условие не выполняется (*условие 1* ложно), то последовательно оценивается каждое условие ELSEIF. При выполнении этого условия используются операторы данного блока. Если ни одно из условий ELSEIF не выполнено, то подключаются операторы блока ELSE.

Блоки ELSEIF и ELSE могут быть опущены. В блочную структуру оператора IF...THEN можно вставить любое количество условий ELSEIF. Каждый из блоков может также содержать вложенные блочные структуры.

Блок должен заканчиваться ключевым словом END IF. Ключевые слова IF, ELSEIF, ELSE и END IF должны быть первыми в строке. Структура считается блочной, если в блоке IF после THEN в той же строке операторов нет (кроме комментариев).

Поясним вышесказанное на примерах.

Пример 5.2. Блочная форма оператора *IF...THEN*

Пусть требуется реализовать следующее ветвление (рис. 5.7):

$$\begin{cases} x < 0, & \text{то } y = x; \\ x = 0, & \text{то } y = 5^x; \\ 0 < x < 5, & \text{то } y = 2x; \\ x \geq 5, & \text{то } y = x^2. \end{cases}$$

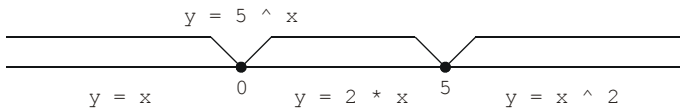


Рис. 5.7. К условию примера 5.2

Программа 5.4

```
'5B-1uOp.bas
10 CLS   'очистка экрана
20 PRINT "Программа с разветвлениями, вариант 100"
30 PRINT "Выход — клавиша Esc, продолжение — любая клавиша"
40 PRINT
50 INPUT "x = ", x
60 IF x < 0 THEN
     y = x
     ELSEIF x = 0 THEN
     y = 5 ^ x
     ELSEIF x => 5 THEN
     y = x ^ 2
     ELSE
     y = 2 * x
     END IF
70 PRINT "y = "; y
80 SLEEP
90 IF INKEY$ = CHR$(27) THEN  ELSE 40
100 END
```

Операторы `PRINT` в строках 20—40, 70 обеспечивают необходимый интерфейс программы 5.4. С помощью оператора `INPUT` в строке 50 вводится значение `x`, обрабатываемое затем оператором `IF...THEN` в блочной форме в строке 60. Оператор `IF...THEN` в строке 90 обеспечивает завершение программы в случае нажатия клавиши <Esc>. При нажатии другой клавиши программа предлагает ввести новое значение `x`. Оператор `SLEEP` в строке 80 реализует паузу в выполнении программы до нажатия какой-либо клавиши.

Результат работы программы 5.4 дан на рис. 5.8.

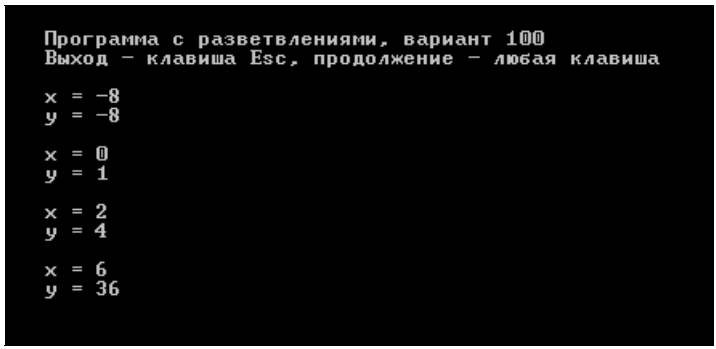


Рис. 5.8. Пример работы с блочной формой оператора `IF...THEN`

Пример 5.3. Рисование ломаной линии

Требуется разработать программу рисования ломаной линии.

Программа 5.5

```
'5B-2uOp.bas
10 CLS        'очистка экрана
20 SCREEN 9   'установка экранного режима
30 y = 30
40 dy = 0
50 FOR x = 5 TO 620
60 IF x >= 50 AND x < 150 THEN
     dy = .5
     ELSEIF x => 150 AND x < 250 THEN
     dy = -.5
      ELSEIF x => 250 AND x < 450 THEN
     dy = .2
     ELSEIF x => 450 THEN
     dy = -.2
     END IF
70 CIRCLE (x, y), 2, 15
80 PAINT STEP(0, 0), 15, 15
90 FOR iv = 1 TO 2000: NEXT iv      'выдержка времени
100 y = y + dy
110 NEXT x
120 END
```

Отображение ломаной линии осуществляется перемещающейся окружностью радиуса 2, рисуемой оператором `CIRCLE` в строке 70 и закрашиваемой оператором `PAINT` в строке 80. Очевидно, что вид излома определяется величиной `dy` в строке 100. Движение окружности задает цикл `FOR...NEXT` в строках 50—110. Начальное значение `dy` для строки 100 задается в строке 40, а затем `dy` изменяется с помощью оператора `IF...THEN` в блочной форме в строке 60. Выдержка времени в строке 90 позволяет наблюдать процесс рисования.

Результат работы программы 5.5 показан на рис. 5.9.

Рис. 5.9. Программа рисования ломаной линии

5.2. Оператор *SELECT CASE*

SELECT CASE — управляющий оператор, выполняющий один или несколько блоков операторов в зависимости от значения выражения выбора. В общем виде оператор SELECT CASE выглядит следующим образом:

```
SELECT CASE (выражение выбора)
CASE (список выражений 1)
    (операторы блока 1)
CASE (список выражений 2)
    (операторы блока 2)
CASE (список выражений 3)
    (операторы блока 3)
 ...
CASE ELSE
    (операторы блока N)
END SELECT
```

Здесь *выражение выбора* — любое числовое или символьное выражение (например, k — числовое или p$ — символьное).

Список выражений — одно или более выражений того же типа, что и *выражение выбора*. Списку выражений предшествует ключевое слово CASE.

Операторы_блока содержат любое количество операторов.

Если несколько конкретизировать для числового выражения выбора, то оператор SELECT CASE примет следующий вид:

```
SELECT CASE k
CASE 1
    (операторы блока 1)
CASE 2, 3, 4
    (операторы блока 2 3 4)
CASE 5 TO 7
    (операторы блока 5-7)
CASE IS > 7
    (операторы блока 8)
 ...
CASE ELSE
    (операторы блока N)
END SELECT
```

Здесь приведены примеры всех форм списка выражений. В этом списке может быть:

- лишь одно выражение (CASE 1);
- несколько выражений, перечисленных через запятую (CASE 2, 3, 4);

- несколько выражений, заданных с помощью ключевого слова TO (CASE 5 TO 7). Отметим, что CASE 5 TO 7 эквивалентно CASE 5, 6, 7;
- несколько выражений, заданных с помощью ключевого слова IS и операторов отношения из табл. 5.1 (например, CASE IS > 7).

Если выражение выбора отвечает условиям списка выражений данного блока CASE, то выполняются операторы из этого блока. После этого управление передается оператору, следующему в программе за END SELECT.

Если используется ключевое слово TO для определения пределов выражения, то меньшее значение должно быть первым. Например, операторы блока CASE –2 TO –4 не выполняются, если выражение выбора равно –3. Для того чтобы они выполнялись при –3, эта строка должна быть записана как CASE –4 TO –2.

Операторы сравнения можно использовать только с ключевым словом IS.

Блок операторов CASE ELSE выполняется только в том случае, если выражение выбора не удовлетворяет ни одному из условий CASE. Обычно используется для обработки нежелательных значений.

Можно использовать несколько выражений или пределов в каждом условии CASE, например:

```
CASE 1 TO 5, 6 TO 8, 9, 10, IS > 11
```

Это же справедливо и для символьных выражений:

```
CASE "авиация", p$, "ББВ" TO "ТТТ"
```

В этом случае CASE выбирает строки выражения выбора, которые равны значению "авиация", текущему значению символьной переменной p$ или находятся между "ББВ" и "ТТТ" в алфавитном порядке. Строки оцениваются в соответствии с ASCII-кодами их символов. Так строчные буквы имеют бóльшие ASCII-коды, чем прописные. Например:

```
"мозг" > "Мозг" > "МОЗГ"
```

Если выражение выбора удовлетворяет нескольким условиям CASE, то выполняется блок операторов, идущий первым. Блоки оператора SELECT CASE могут быть вложенными. Каждый вложенный блок должен быть завершен END SELECT.

Рассмотрим особенности использования условного оператора SELECT CASE на примерах.

Пример 5.4. Перебор вариантов с помощью оператора *SELECT CASE*

Разработать программу, выводящую расписание приема врачей.

Программа 5.6

```
'5SC-1uOp.bas
10 CLS              'очистка экрана
20 PRINT SPC(5); "Расписание по кабинетам"
30 PRINT SPC(3); "Время приема:"
40 PRINT SPC(5); "9 час. 00 мин. — 14 час. 00 мин."
50 PRINT
60 FOR n = 1 TO 6
70 SELECT CASE n
     CASE 1: c$ = "Главврач"
     CASE 2: c$ = "Хирург"
```

```
     CASE 3: c$ = "Невропатолог"
     CASE 4, 5, 6: c$ = "Терапевт": a = 1
80 END SELECT
90 PRINT SPC(6); "Кабинет"; n; "  "; c$;
100 IF a = 1 THEN PRINT n - 3; "-го участка";
110 PRINT
120 NEXT n
130 END
```

Результат работы программы 5.6 приведен на рис. 5.10.

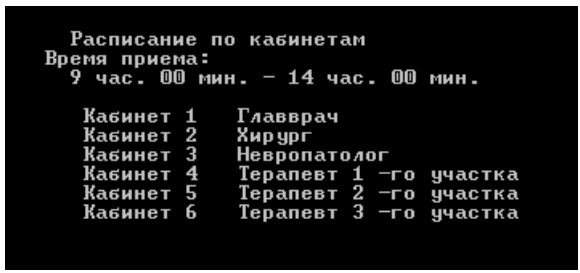


Рис. 5.10. Программа перебора вариантов с помощью оператора SELECT CASE

Программа 5.6 осуществляет перебор шести вариантов с помощью цикла FOR...NEXT в строках 60—120 и оператора SELECT CASE в строке 70. Операторы в строках 20—50 и 90—110 обеспечивают интерфейс программы, причем строка 100 служит для вывода номера участка в случае терапевта (для этого используется переключатель a). Для вывода номера участка в той же строке, что и терапевт, в конце строки 90 поставлена точка с запятой. В данной программе в списке выражений после CASE используются варианты лишь с одним выражением и несколькими, перечисленными через запятую.

В следующем примере для задания пределов выражения применены ключевые слова TO и IS.

Пример 5.5. Использование *TO* и *IS* в операторе *SELECT CASE*

Разработать программу, выводящую физическое состояние воды в зависимости от температуры.

Программа 5.7

```
'5SC-2uOp.bas      Состояние воды
10 CLS             'очистка экрана
20 PRINT "Выход - клавиша Esc, продолжение - любая клавиша"
30 PRINT
40 INPUT "Температура t = ", t
50 SELECT CASE t
     CASE IS <= 0: v$ = "лед"
     CASE IS => 100: v$ = "пар"
     CASE 1 TO 99: v$ = "жидкость"
60 END SELECT
70 PRINT " Состояние воды - "; v$
```

```
80 SLEEP
90 IF INKEY$ = CHR$(27) THEN  ELSE 30
100 END
```

Результат работы программы 5.7 приведен на рис. 5.11.

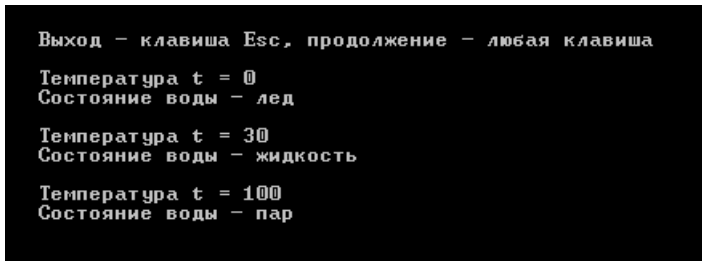


Рис. 5.11. Программа с использованием ключевых слов `TO` и `IS` в операторе `SELECT CASE`

Программа 5.7 простая и прозрачная и практически не требует комментариев. Отметим лишь, что строки 80—90 позволяют вывести на экран несколько вариантов без перезапуска программы.

В программах 5.6 и 5.7 использовались числовые выражения выбора, а, между тем, выражение выбора может быть и символьным (строковым).

Пример 5.6. Символьное выражение выбора в операторе *SELECT CASE*

Разработать программу, выводящую информацию о крупнейших озерах России.

Программа 5.8

```
'5SC-3uOp.bas      Озера России
10 CLS             'очистка экрана
20 PRINT "   Крупные озера России"
30 PRINT
40 PRINT " Байкал, Ладожское, Онежское"
50 PRINT " Таймыр, Ханка, Чудское"
60 PRINT
70 INPUT "Какое озеро – ", o$
80 PRINT
90 SELECT CASE o$
     CASE "Байкал":    p = 31500: v = 455: g = 1620
     CASE "Ладожское": p = 17700: v = 5:   g = 230
     CASE "Онежское":  p = 9720:  v = 33:  g = 127
     CASE "Таймыр":  p = 4560:    v = 6:   g = 26
     CASE "Ханка":   p = 4190:    v = 68:  g = 11
     CASE "Чудское": p = 3550:    v = 30:  g = 15
100 END SELECT
110 PRINT "Площадь"; p; "кв.км"
120 PRINT "Высота над уровнем моря"; v; "м"
130 PRINT "Наибольшая глубина"; g; "м"
140 END
```

Результат работы программы 5.8 дан на рис. 5.12.

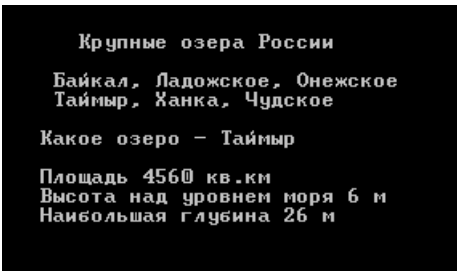


Рис. 5.12. Программа с использованием символьного выражения выбора в операторе SELECT CASE

Операторы PRINT в строках 40—50 выводят на экран своеобразное меню-подсказку, помогающее выбрать озеро. Выбор осуществляется посредством оператора INPUT в строке 70 в символьную переменную o$ (на что указывает суффикс $). Причем слово должно начинаться с большой буквы и должно быть написано так, как в подсказке, или аналогично тому, как указано после CASE.

Числовое выражение выбора может быть и отрицательным.

Пример 5.7. Использования оператора *SELECT CASE* при создании движения

Разработать программу, рисующую спортсмена, делающего упражнение — махи руками.

Программа 5.9

```
'5SC-4uOp.bas      Зарядка. Махи руками.
10 CLS             'очистка экрана
20 SCREEN 9   'установка экранного режима
'==1===== Блок движения ==========================================
30 ruki = 1
40 x = 300
50 y = 100
60 DO
70 GOSUB 140                          'рисует спортсмена
80 FOR iv = 1 TO 300000: NEXT iv      'выдержка времени
90 IF INKEY$ = CHR$(27) THEN EXIT DO  'выход из цикла — клавиша <Esc>
100 CLS                               'стирает спортсмена
110 ruki = - ruki
120 LOOP
'==2===== Подпрограмма рисования спортсмена ======================
130 GOTO 280
140 CIRCLE (x, y), 10, 14
150 PAINT STEP(0, 0), 14, 14
160 CIRCLE STEP(0, 16), 11, 12
170 PAINT STEP(0, 0), 12, 12
180 LINE STEP(-11, 0) - STEP(22, 22), 12, BF
190 DRAW "C9 NL22 M+2,+12 L11 H2 G2 L11 NM+2,-12 BE2 P9,9"
200 DRAW "BG2 C14 R10 M-1,+30 L7 NM-2,-30 BE2 P14,14 BG2"
210 LINE  - STEP(7, 3), 9, BF
220 LINE STEP(8, 0) - STEP(7, -3), 9, BF
```

```
230 DRAW "C14 NL7 M+2,-30 L11 NM+1,+30 BF2 P14,14"
240 PSET (x — 11 * ruki, y + 37), 12
250 SELECT CASE ruki
      CASE -1
      DRAW "M-15,-12 u2 M+20,-12 D8 M-10,+6 NM+6,+7 BL5 P12,12"
      DRAW "BM+29,-14 M+8,-14 U14 R7 D18 NM-12,+20 BL3 P12,12"
      DRAW "C14 BM+2,-19 L5 H1 U5 E2 R2 F2 D5 G1 BH2 P14,14"
     CASE 1
      DRAW "M+15,-12 u2 M-20,-12 D8 M+10,+6 NM-6,+7 BR5 P12,12"
      DRAW "BM-29,-14 M-8,-14 U14 L7 D18 NM+12,+20 BR3 P12,12"
      DRAW "C14 BM+3,-19 L5 H1 U5 E2 R2 F2 D5 G1 BH2 P14,14"
260 END SELECT
270 RETURN
'================================================================
280 END
```

Результат работы программы 5.9 приведен на рис. 5.13.

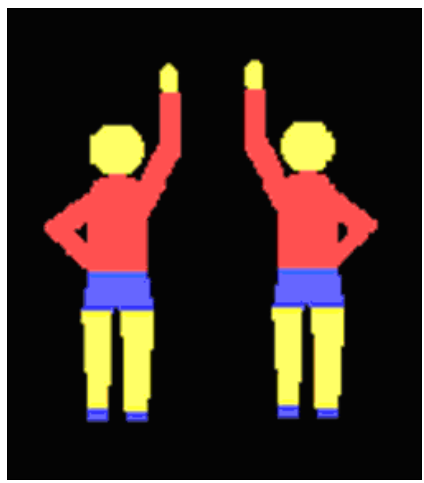


Рис. 5.13. Использования оператора `SELECT CASE` при создании движения

Программа 5.9 рисует фигуру спортсмена с поочередно меняющимся положением рук. Одно положение: левая рука — вверх, правая — на поясе; другое: правая рука — вверх, левая — на поясе. Смену положения рук обеспечивает строка 110 (меняет значение `ruki` с –1 на 1 и наоборот) и оператор `SELECT CASE` в строке 250, причем оператор `PSET` в строке 240 предназначен для привязки начала рисования рук к начальным координатам рисования фигуры спортсмена. В блоке движения программы организован цикл `DO...LOOP`, в теле которого происходит обращение с помощью оператора `GOSUB` в строке 70 к подпрограмме рисования спортсмена в строках 140—270. После выдержки времени посредством пустого цикла `FOR...NEXT` в строке 80 изображение спортсмена стирается оператором `CLS` в строке 100. С новым значением `ruki` происходит очередное обращение к подпрограмме рисования (`GOSUB` в строке 70) и т. д. В подпрограмме рисования спортсмена операторы в строках 140—150 рисуют голову, операторы в строках 160—180 — туловище, в строке 190 — трусы, в строках 200 и 230 — ноги, в строках 210—220 — кроссовки. Блок операторов в `CASE –1` оператора `SELECT CASE` в строке 250 рисуют одно положение рук, а в `CASE 1` — другое. Для изме-

нения темпа махов руками следует соответственно увеличить или уменьшить число повторений в пустом цикле FOR...NEXT в строке 80.

Впрочем, если кому-либо не нравится отрицательное значение выражения выбора, то можно перейти к обычным CASE 1 и CASE 2. Но тогда строка 110 программы 5.9 будет выглядеть следующим образом:

```
110 IF ruki = 1 THEN ruki = 2 ELSE ruki = 1
```

5.3. Оператор безусловного перехода *GOTO*

Ветвление в программах наряду с условными операторами можно осуществлять и посредством оператора безусловного перехода GOTO.

Оператор имеет следующий вид:

```
GOTO m
```

где m — метка.

Ветвление с помощью оператора безусловного перехода может быть организовано, например, так:

```
 ...
GOTO m
 ...
m:
PRINT "Завершение программы"
END
```

Обратите внимание, что метка снабжена двоеточием. Но чаще в качестве метки используется номер линии, после которого двоеточие не ставится:

```
 ...
50 GOTO 100
 ...
100 PRINT "Завершение программы"
110 END
```

Многие авторы настойчиво рекомендуют не использовать в своих программах оператор GOTO, поскольку его использование, по их мнению, ломает логический строй программы. Они предлагают разнообразные приемы, лишь бы избежать GOTO. Авторы же этой книги считают, что уместное использование оператора безусловного перехода не портит программы. Например:

```
 ...
50 IF 0 < x THEN a = 1: S = S + 1: GOTO 120
 ...
120 PRINT "S = "; S
130 END
```

Можно привести и пример неуместного использования оператора безусловного перехода:

```
 ...
50 IF 0 < x THEN GOTO 120
 ...
120 PRINT "y = "; 2 * x + 1: GOTO 230
 ...
230 END
```

Гораздо логичнее была бы исключить строку 120, изменив строку 50

```
50 IF 0 < x THEN PRINT "y = "; 2 * x + 1: GOTO 230
```

Также неуместным представляется создание с помощью GOTO циклов, т. к. есть специальные операторы цикла FOR...NEXT, DO...LOOP и WHILE...WEND. Вот пример такого неуместного использования:

```
10 CLS
20 P = 1
30 N = 1
40 IF N > 8 THEN 80
50 P = P * N
60 N = N + 1
70 GOTO 40
80 PRINT "P = "; P
90 END
```

Логичнее при разработке подобной программы применить специально для этого созданный оператор цикла FOR...NEXT.

Итак, резюмируя вышесказанное, можно посоветовать: если вам удобно использовать оператор GOTO, то используйте его. Но не нарушайте при этом логику программы.

Ветвление в программах можно осуществлять посредством оператора ON...GOTO. В общем виде он выглядит следующим образом:

```
ON a GOTO N1, N2, ... , Nk
```

где a — арифметическое выражение или переменная, N1, N2, ... , Nk — список номеров линий.

Логика работы этого оператора следующая: рассматривается значение выражения a. Если a = k, то происходит переход к строке с номером Nk. Если же значение a не входит в диапазон [1;k], происходит переход на следующую после оператора ON...GOTO строку программы. Если значение a — действительное число, то оно округляется до целого числа.

Пример 5.8. Использование оператора *ON...GOTO*

Пусть билеты в театр, мюзик-холл, цирк стоят 300, 500 и 400 рублей соответственно. Разработать программу, запрашивающую, куда желает пойти пользователь и количество билетов. После чего выводится стоимость заказанных билетов.

Программа 5.10

```
'5G-1uOp.bas      Театральная касса
10 CLS            'очистка экрана
20 PRINT "Театр — 1, Мюзик-холл — 2, Цирк — 3"
30 PRINT
40 INPUT "Куда идете — ", nu
50 INPUT "Сколько билетов — ", kb
60 ON nu GOTO 70, 80, 90
70 cb = 300: GOTO 100
80 cb = 500: GOTO 100
90 cb = 400: GOTO 100
100 PRINT
110 PRINT "С Вас — "; cb * kb; "рублей"
120 END
```

Оператор PRINT в строке 20 выводит на экран своеобразное меню-подсказку для выбора развлекательного учреждения. Операторы INPUT в строках 40—50 обеспечивают ввод входных данных. Причем значение переменной nu изменяется в диапазоне от 1 до 3. Если nu = 1, оператор в строке 60 задает переход на строку 70; если nu = 2, то на строку 80; если nu = 3, то на строку 90. Оператор PRINT в строке 110 обеспечивает вывод результата. Операторы GOTO в строках 70—90 необходимы для перехода на строку 110 после присваивания соответствующего значения цены билета.

Результат работы программы 5.10 приведен на рис. 5.14.

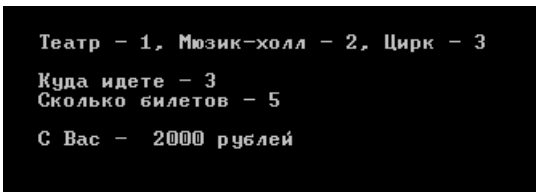
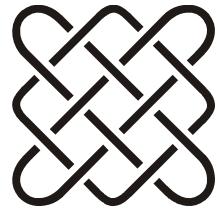


Рис. 5.14. Программа "Театральная касса": 5 человек желают пойти в цирк



ГЛАВА 6

Операторы цикла

6.1. Назначение циклов

Операторы циклов существуют во всех языках программирования и являются мощным инструментом для создания красивых и эффективных программ. Покажем это на примере, разработав программу без применения цикла и программу, решающую ту же самую задачу, но с использованием цикла.

Пример 6.1. Построение графика по точкам

Требуется построить график функции по точкам: $x = 80$ и $y = 160$; $x = 120$ и $y = 167$; $x = 160$ и $y = 172$; $x = 200$ и $y = 175$; $x = 240$ и $y = 177$; $x = 280$ и $y = 178$; $x = 320$ и $y = 178$.

Программа 6.1

```
'6-0cikl.bas
10 CLS        'очистка экрана
20 SCREEN 9
'==1===== Блок рисования осей координат ============================
30 LINE (0, 250) — (640, 250)        'Ось абсцисс (x)
40 LINE (120, 0) — (120, 350)        'Ось ординат (y)
'==2===== Блок рисования кривой ====================================
50 x1 = 200
60 y1 = 90
70 LINE (x1, y1) — (x1, 250), , , &H8888
'--------------------------------------------------------------------
80 x2 = 240
90 y2 = 83
100 LINE (x2, y2) — (x2, 250), , , &H8888
110 LINE (x1, y1) — (x2, y2)
'--------------------------------------------------------------------
120 x3 = 280
130 y3 = 78
140 LINE (x3, y3) — (x3, 250), , , &H8888
150 LINE (x2, y2) — (x3, y3)
'--------------------------------------------------------------------
160 x4 = 320
170 y4 = 75
```

```
180 LINE (x4, y4) — (x4, 250), , , &H8888
190 LINE (x3, y3) — (x4, y4)
'-----------------------------------------------------------------
200 x5 = 360
210 y5 = 73
220 LINE (x5, y5) — (x5, 250), , , &H8888
230 LINE (x4, y4) — (x5, y5)
'-----------------------------------------------------------------
240 x6 = 400
250 y6 = 72
260 LINE (x6, y6) — (x6, 250), , , &H8888
270 LINE (x5, y5) — (x6, y6)
'-----------------------------------------------------------------
280 x7 = 360
290 y7 = 73
300 LINE (x6, y6) — (x6, 250), , , &H8888
310 LINE (x6, y6) — (x7, y7)
'=================================================================
320 END
```

Отметим, что пунктирный стиль линии в операторе `LINE` в строках 70, 100, 140, 180, 220, 260 и 300 задается шестнадцатеричным числом 8888 (суффикс `&H` — показатель шестнадцатеричного числа).

Результат работы программы 6.1 представлен на рис. 6.1.

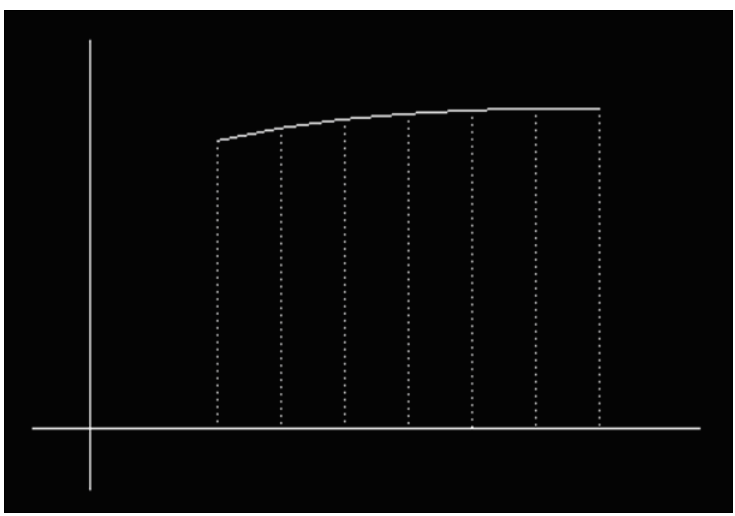


Рис. 6.1. Программа построения графика по точкам

Анализируя программу 6.1, нетрудно заметить, что блок операторов, состоящий из операторов присваивания и операторов `LINE`, повторяется 7 раз, что приводит к неоправданному удлинению программы. Именно для таких случаев и предназначены операторы цикла. Посмотрим, как изменится программа, решающая ту же самую задачу построения графика по точкам, с использованием операторов цикла.

Программа 6.2

```
'6-1cikl.bas
10 CLS        'очистка экрана
20 SCREEN 9
'==1===== Блок рисования осей координат ============================
30 LINE (0, 250) — (640, 250)        'Ось абсцисс (x)
40 LINE (120, 0) — (120, 350)        'Ось ординат (y)
'==2===== Блок рисования кривой ====================================
50 DATA 90,83,78,75,73,72,72
60 FOR x = 200 TO 440 STEP 40
70 i = (x — 160) / 40
80 READ y(i)
90 LINE (x, y(i)) — (x, 250) , , , &H8888
100 IF x > 200 THEN LINE (x — 40, y(i — 1)) — (x, y(i))
110 NEXT x
'====================================================================
120 END
```

Результат работы программы 6.2 будет таким же, как и программы 6.1 (рис. 6.1).

Строки 10—40 программ 6.1 и 6.2 совпадают. В отличие от программы 6.1, в которой значения координат точек вводились посредством операторов присваивания, в программе 6.2 для этого используется пара операторов `DATA/READ` в строках 50, 80, а `x` задается с помощью оператора цикла `FOR...NEXT`, организованного в строках 60—110. Причем в тело цикла входят многократно повторяющиеся в программе 6.1 операторы `LINE`. Строка 70 нужна для перевода значений `x` в значения `i` = 1, 2, 3 ... , 7.

Нетрудно заметить, что использование цикла не только сократило длину программы почти втрое, но и сделало ее более красивой. Надеемся, что данный пример наглядно показал преимущества использования циклов в подобных случаях.

6.2. Оператор *FOR...NEXT*

Имеется очень много задач, в которых заранее известно или легко вычислить число повторений цикла. Для решения таких задач и предназначен оператор цикла `FOR...NEXT` — управляющий оператор, повторяющий блок операторов указанное число раз. В общем виде оператор цикла выглядит следующим образом:

```
FOR I = In TO Ik STEP p
  (тело цикла)
NEXT I
```

где `I` — счетчик цикла; `In` — начальное значение счетчика; `Ik` — конечное значение счетчика; `p` — шаг.

Значения `In`, `Ik`, `p` указываются в первой строке цикла, например:

```
FOR I = 2 TO 8 STEP 2
```

или до начала выполнения цикла, например:

```
In = 2: Ik = 8: p = 2
FOR I = In TO Ik STEP p
```

В качестве параметров начала, конца и счетчика цикла FOR...NEXT по умолчанию поддерживаются значения двойной точности, но предпочтительнее использовать целые переменные.

Шаг может быть положительным или отрицательным. Как правило, он выражается целым числом, но может быть и дробным.

Когда шаг положительный, то цикл будет выполняться, если начальное значение счетчика не превышает конечное значение. При отрицательном шаге для выполнения цикла конечное значение должно быть меньше или равно начальному. Иными словами, условие выполнения цикла FOR...NEXT выглядит следующим образом:

- при $p > 0$ должно выполняться $Ik \geq In$;
- при $p < 0$ должно выполняться $In \geq Ik$.

При любых значениях шага p цикл проработает только один раз, если начальное и конечное значения счетчика цикла совпадают, т. е. если $In = Ik$.

По умолчанию шаг равен 1, в этом случае первая строка цикла обычно записывается без служебного слова STEP:

```
FOR I = In TO Ik
```

Между ключевыми словами цикла располагается повторяющийся указанное число раз блок операторов, называемый телом цикла.

После ключевого слова NEXT счетчик цикла I может отсутствовать, однако не рекомендуется прибегать к такому упрощению программы, особенно при использовании вложенных циклов.

Логика цикла FOR...NEXT представлена при разных значениях шага блок-схемами на рис. 6.2.

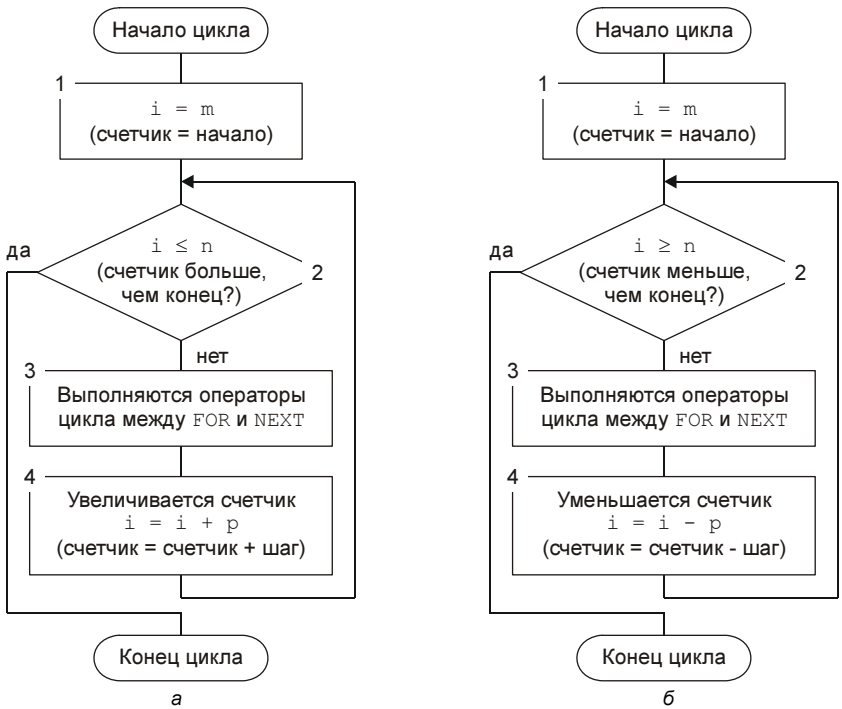


Рис. 6.2. Логика цикла FOR...NEXT: при положительном значении шага (*а*); при отрицательном значении шага (*б*)

Пример 6.2. Вычисление значений функции

Требуется вычислить значения функции $y = ax^2 + bx + c$ в диапазоне от 4 до 20 с шагом 2.

Программа 6.3

```
'6F-2Cikl.bas
10 CLS        'очистка экрана
20 PRINT "Вычислить значения функции y = a*x^2 + b*x + c в диапазоне
                  от 4 до 20 с шагом 2"
30 PRINT
40 PRINT "Задайте значения коэффициентов a, b, c"
50 INPUT " a = ", a
60 INPUT " b = ", b
70 INPUT " c = ", c
80 PRINT SPC(7); "Значения аргумента и функции"
90 FOR x = 4 TO 20 STEP 2
100 y = a * x ^ 2 + b * x + c
110 PRINT TAB(12); "x = "; x; TAB(23); "y = "; y
120 NEXT x
130 END
```

Строки 20—80 обеспечивают интерфейс программы: PRINT в строке 20 выводит на экран условие задачи, PRINT в строке 30 разделяет выводимые тексты, операторы в строках 40—70 обеспечивают ввод коэффициентов вычисляемой функции, причем функция SPC при PRINT в строке 80 с заданным числом пробелов, равным 7, позиционирует выводимую надпись. Сам цикл FOR...NEXT организован в строках 90—120. Операторы в строках 100—110 составляют тело цикла, которое выполняется 9 раз с разными значениями параметра (счетчика) цикла x, который изменяется от начального значения, равного 4, до конечного значения, равного 20, с приращением (шагом) 2.

На первом шаге цикла x = 4, на втором x = 4 + 2 = 6. Каждый шаг цикла увеличивает значение параметра x на величину приращения, равного 2. На последнем девятом шаге будет, наконец, выполнено вычисление y и для конечного значения параметра цикла, равного 20. На каждом шаге цикла в строке 100 вычисляется значение y, а PRINT в строке 110 обеспечивает попарный вывод значений x и y, причем функции TAB при операторе PRINT в строке 110 определяют позиции вывода. Результат работы программы 6.3 приведен на рис. 6.3.

Если в программе по каким-либо причинам переменная x или y получит значение, выходящее за границы интервала назначенного ей типа, произойдет немедленный останов программы, и интерпретатор сообщит о том, что произошло переполнение — Overflow.

Существует много задач, при решении которых необходимо использовать циклы с отрицательным шагом, например, когда нужно организовать обратный отсчет. Такая же задача стоит при реализации алгоритма перевода числа из десятичного в другое счисление, скажем в двоичное. Там цифры в результате целочисленного деления появляются и записываются в память в обратном порядке. Поэтому для вывода полученного двоичного числа требуется изменить порядок следования цифр на противоположный. Как решаются подобные задачи, можно проследить на следующем примере.

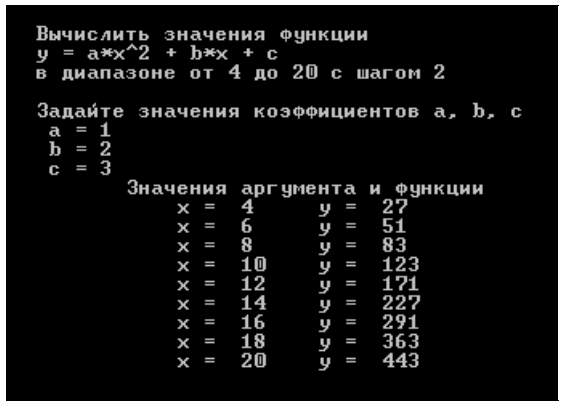


Рис. 6.3. Программа вычисления значений функции

Пример 6.3. Вывод массива в обратном порядке

Требуется вывести элементы массива в обратном порядке.

Программа 6.4

```
'6F-3cikl.bas
10 CLS        'очистка экрана
20 PRINT "Вывести элементы массива в обратном порядке"
30 PRINT
40 INPUT "Задайте размер массива n = ", n
50 DIM x(n)
'==1=== Блок ввода и вывода массива ================================
60 PRINT "Введенный массив"
70 FOR i = 1 TO n
80 x(i) = i
90 PRINT x(i);
100 NEXT i
110 PRINT
'==2=== Блок заданного вывода массива ==============================
120 PRINT "Заданный вывод массива"
130 FOR j = n TO 1 STEP -1
140 PRINT x(j);
150 NEXT j
'====================================================================
160 END
```

Оператор `PRINT` в строке 20 обеспечивает вывод на экран краткой формулировки поставленной задачи, а `PRINT` в строке 30 служит для визуального разделения выводимых текстов. Оператор `DIM` в строке 50 подготавливает в памяти место для вводимого массива, размер которого определяется в строке 40 с помощью оператора `INPUT`. Цикл с шагом 1 в строках 70—110 (строки 80—90 — тело цикла) обеспечивает ввод и вывод массива, причем для наглядности элементами массива являются натуральные числа от 1 до `n` (это обеспечивает оператор присваивания в строке 80). Точка с запятой после оператора `PRINT` в строке 90

определяет вывод элементов массива в одну строку, обрываемую оператором PRINT в строке 110. Обратный вывод элементов массива осуществляется циклом в строках 130—150.

Результат работы программы 6.4 приведен на рис. 6.4.

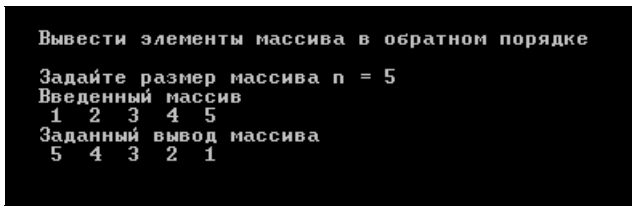


Рис. 6.4. Программа вывода массива

Иногда выгоднее использовать один и тот же цикл, но с разными параметрами. Для уже рассмотренного примера 6.3 такой прием приводится в программе 6.5, являющейся модифицированным вариантом программы 6.4. Поэтому результат работы программы 6.5 будет аналогичен результату работы программы 6.4, приведенному на рис. 6.4.

Программа 6.5

```
'6F-4cikl.bas
10 CLS        'очистка экрана
20 PRINT "Вывести элементы массива в обратном порядке"
30 PRINT
40 INPUT "Задайте размер массива n = ", n
50 DIM x(n)
'==1=== Блок ввода и вывода массива ================================
60 PRINT "Введенный массив"
70 In = 1: Ik = n: p = 1
80 FOR i = In TO Ik STEP p
90 x(i) = i
100 PRINT x(i);
110 NEXT i
120 PRINT
'==2=== Блок изменения параметров ==================================
130 IF e = 0 THEN e = 1: In = n: Ik = 1: p = -1:
                    PRINT "Заданный вывод массива": GOTO 80
'====================================================================
140 END
```

Параметры счетчика цикла в строке 80 даются в общем виде, а их конкретные значения определяются в строке 70 для прямого вывода массива и в строке 130 для обратного вывода. В строке 130 используется переключатель e, позволяющий избежать зацикливания из-за GOTO в этой строке.

Поскольку величина шага в цикле может быть произвольной, а не только 1 или –1, как в некоторых других языках программирования, то появляется вариантность при разработке программ. Поясним это на примере 6.4.

Пример 6.4. Вывод цветной ленты

Требуется разработать программу, выводящую ленту, состоящую из квадратов разного цвета.

Программа 6.6

```
'6F-5cikl.bas
10 CLS          'очистка экрана
20 SCREEN 9       'установка экранного режима
30 COLOR 14       'установка экранных цветов
40 FOR i = 1 TO 5
50 x = 100 + 10 * (i — 1)
60 LINE (x, 100)-STEP(10, 10), i, BF
70 NEXT i
80 END
```

Программа 6.7

```
'6F-6cikl.bas
10 CLS          'очистка экрана
20 SCREEN 9       'установка экранного режима
30 COLOR 14       'установка экранных цветов
40 FOR x = 100 TO 140 STEP 10
50 cv = (x — 90) / 10
60 LINE (x, 100)-STEP(10, 10), cv, BF
70 NEXT x
80 END
```

Результат работы программ 6.6 и 6.7 приведен на рис. 6.5.

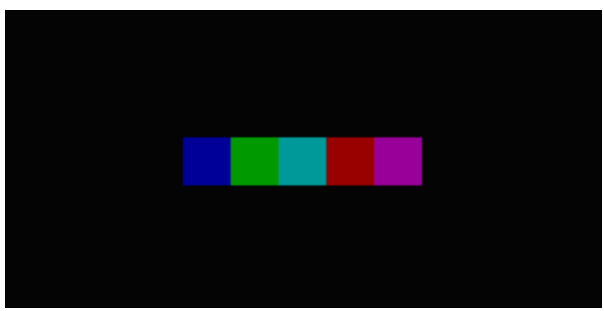


Рис. 6.5. Программа вывода цветной ленты

В программе 6.6 цикл реализован по `i` с шагом 1. Цвет в операторе `LINE` в строке 60 изменяется по шагам цикла от 1 до 5. В строке 50 стоит выражение для `x`, обеспечивающее его изменение от 100 до 140 с шагом 10. В программе 6.7 цикл реализован по `x` с шагом 10, а выражение в строке 50 задает изменение цвета в операторе `LINE` в строке 60 от 1 до 5. Результат же работы этих программ одинаков.

С помощью циклов удобно вычислять суммы.

Пример 6.5. Вычисление суммы

Найти сумму семи значений функции $y = ax^2 + bx$ при $x = 1, 2, \ldots, 7$, выводя полученное значение суммы на каждом шаге.

Программа 6.8

```
'6F-7cikl.bas
10 CLS
20 PRINT "Вычисление суммы"
30 PRINT
40 PRINT "Задайте значения коэффициентов a, b"
50 INPUT "a =", a
60 INPUT "b =", b
70 PRINT SPC(7); "Значения суммы"
80 FOR x = 1 TO 7
90 S = S + (a * x ^ 2 + b * x)
100 PRINT SPC(4); "На"; x; "-ом шаге сумма S ="; S
110 NEXT x
120 END
```

Операторы в строках 20—40, 70, 100 обеспечивают интерфейс программы, причем `SPC` при `PRINT` в строках 70, 100 позиционирует вывод сообщений. Операторы `INPUT` в строках 50—60 служат для ввода значений коэффициентов функции. Оператор цикла `FOR...NEXT` в строках 80, 110 обеспечивает вычисление суммы (строка 90) и вывод ее значений на экран (строка 100).

Результат работы программы 6.8 представлен на рис. 6.6.

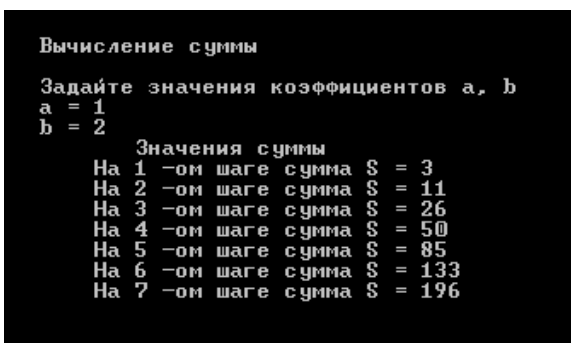


Рис. 6.6. Программа вычисления суммы

Следует обратить внимание, что суммирование в строке 90 происходит вследствие наличия `S` справа от оператора присваивания. Действительно, если коэффициенты a = 1 и b = 2, то

1-й шаг: `S` = 0, `x` = 1, тогда `S` = 0 + 1^2 + 2 * 1 = 3;

2-й шаг: `S` = 3, `x` = 2, тогда `S` = 3 + 2^2 + 2 * 2 = 11;

3-й шаг: `S` = 11, `x` = 3, тогда `S` = 11 + 3^2 +2 * 3 = 26;

...

7-й шаг: `S` = 133, `x` = 7, тогда `S` = 133 + 7^2 + 2 * 7 = 196,

т. е. на 7-м шаге:

S = S(x = 0) + S(x = 1) + S(x = 2) + ... + S(x = 7) = 196.

Если же в правой части выражения в строке 90 S опустить, то получим:

1-й шаг: x = 1, тогда $S = 1^2 + 2 * 1 = 3$;

2-й шаг: x = 2, тогда $S = 2^2 + 2 * 2 = 8$;

3-й шаг: x = 3, тогда $S = 3^2 + 2 * 3 = 15$,

и т. д.

То есть на каждом шаге цикла вычисляется новое значение функции, которое и присваивается S. Следовательно, суммирования не происходит.

В операторе цикла FOR...NEXT предусмотрена возможность досрочного выхода из цикла (до того, как счетчик цикла достиг своего конечного значения) с помощью EXIT FOR. Такой досрочный выход реализован в программе 6.9.

Программа 6.9

```
'6F-8cikl.bas
10 CLS
20 PRINT "Демонстрация досрочного выхода из цикла"
30 PRINT
40 Xk = 100000
50 t = 20
60 t1 = TIMER
'================================================================
70 FOR x = 1 TO Xk
80 S = S + (2 * x ^ 2 + 3 * x) / x ^ 3
90 t2 = TIMER
100 IF t2 = t1 > t THEN e = 1: EXIT FOR
110 NEXT x
'================================================================
120 IF e = 1 THEN PRINT "Досрочный выход из цикла"
130 PRINT "Сумма S ="; CINT(S)
140 END
```

В программе 6.9 предусмотрен досрочный выход из цикла FOR...NEXT в строках 70—110, если длительность его работы превышает заданный временной интервал. В строке 40 задается число повторений цикла. В строке 50 задается значение допустимого временного интервала в секундах. В строке 60 определяется значение TIMER в момент начала работы цикла. Функция TIMER возвращает число секунд, прошедших с полуночи. В строке 90, входящей в тело цикла, переменной t2 присваивается постоянно меняющееся значение функции TIMER. Следовательно, t2 – t1 определяет время, прошедшее с начала работы цикла. Контроль над временем производится в строке 100 с помощью оператора IF...THEN. Переключатель e необходим, чтобы исключить появление сообщения "Досрочный выход из цикла" в том случае, когда цикл выполнил заданное число повторений и закончил работу.

На рис. 6.7 представлен результат работы программы 6.9.

Допускается вкладывать циклы FOR...NEXT друг в друга, например:

```
10 FOR i =1 TO 5      'это внешний цикл
20 FOR j = 1 TO 5      'это вложенный цикл
```

```
 30 S1 = S1 + a(i , j)    'тело цикла
 40 NEXT j          'сначала закрывается вложенный цикл
 50 NEXT i          'последним закрывается внешний цикл
```

Обычно при использовании циклов `FOR...NEXT` первому счетчику цикла дается имя `i`, вложенному в него — `j`, затем `k`, `l` и далее по алфавиту. Можно для этих целей использовать счетчики с именами `i1`, `i2`, `i3` и т. д.

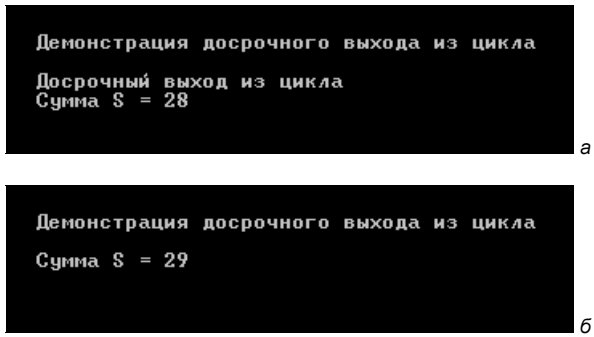


Рис. 6.7. Программа с досрочным выходом из цикла: цикл не уложился в определенный временной интервал (*а*); цикл выполнил заданное число повторений (*б*)

Пример 6.6. Использование вложенных циклов при работе с матрицами

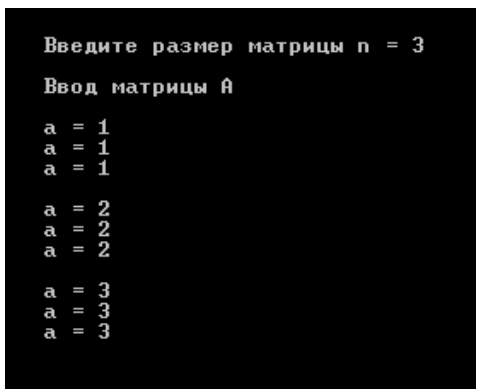
Требуется разработать программу для ввода с помощью оператора `INPUT` и вывода на экран двух квадратных матриц.

Программа 6.10

```
'6F-9cikl.bas
10 CLS
20 INPUT "Введите размер матрицы n = ", n
30 g$ = "Ввод матрицы A"
'==1=== Блок циклов ================================================
40 FOR i = 1 TO 4      'внешний цикл
50 PRINT: PRINT g$: PRINT
     '---2---- Вложенный цикл -------------------------------------
60 FOR j = 1 TO n
     '---3---- Вложенный цикл -------------------------------------
70 FOR k = 1 TO n
80 SELECT CASE i
CASE 1: g$ = "Ввод матрицы B"
               INPUT "a = ", a (j, k)
CASE 2: g$ = "Матрица A"
               INPUT "b = ", b (j, k)
CASE 3: g$ = "Матрица B"
               PRINT a (j, k);
CASE 4: PRINT b (j, k);
90 END SELECT
100 NEXT k
     '-----------------------------------------------------------3----
```

```
110 PRINT
120 NEXT j
     '-----------------------------------------------------------2----
130 IF i <3 THEN CLS
140 NEXT i
'==================================1============================
150 END
```

Программа 6.10 содержит внешний цикл (строки 40—140) и два вложенных цикла (строки 60—120 и 70—100). Иначе говоря, программа содержит тройной цикл, у которого тело цикла содержит двойной вложенный цикл. Причем двойной вложенный цикл служит для работы с матрицами, а внешний — для перебора вариантов этой работы — ввод элементов матрицы A, ввод элементов матрицы B, вывод на экран матрицы A и матрицы B. То есть двойной вложенный цикл используется 4 раза. Для обработки вариантов работы используется условный оператор SELECT CASE в строках 80—90. Конечные значения счетчиков в операторах цикла в строках 60 и 70 задаются в строке 20. Оператор PRINT в строке 50 выводит поочередно сообщения "Ввод матрицы A", "Ввод матрицы B", "Матрица A", "Матрица B", вводимые в строке 30 и в соответствующих CASE оператора SELECT CASE. Оператор CLS в строке 130 стирает с экрана результаты ввода матриц A и B. Результаты работы программы 6.10 даны на рис. 6.8.



а

б

Рис. 6.8. Программа ввода и вывода двух квадратных матриц: ввод элементов матрицы A (*а*); вывод матриц A и B (*б*)

Пример 6.7. Треугольник Паскаля

Требуется разработать программу вывода столбцов треугольника Паскаля. Каждый элемент такого треугольника является суммой двух элементов слева от него.

Программа 6.11

```
'6F-0cikl.bas
10 CLS
20 PRINT "Вывод столбцов треугольника Паскаля"
30 PRINT
40 INPUT "Сколько столбцов хотите вывести (не более 10) — ", Mk
50 DIM e (Mk, Mk)
```

```
'==1===== Блок ввода значений границ ==============================
'Верхняя и нижняя границы = 1
60 FOR i = 1 TO Mk
70 e (i, 1) = 1: e (i, i) = 1
80 NEXT i
'==2===== Блок ввода значений "средних" элементов ==================
90 FOR j = 2 TO Mk
100 FOR k = 2 TO Mk
110 e (j, k) = e (j — 1, k — 1) + e (j — 1, k)
120 NEXT k
125 NEXT j
'==3===== Блок вывода треугольника Паскаля ========================
130 nNy = 13
140 FOR m = 1 TO Mk
150 nx = m + 1
160 ny = nNy
170 FOR n = 1 TO m
180 LOCATE ny, 5 * nx — 9: PRINT e (m, n)
190 ny = ny + 2
200 NEXT n
210 PRINT
220 nNy = nNy — 1
230 NEXT m
'==================================================================
240 END
```

Результат работы программы 6.11 представлен на рис. 6.9.

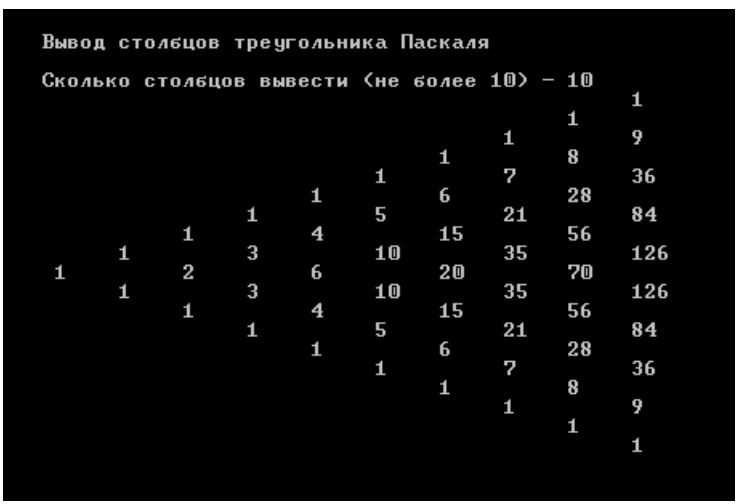


Рис. 6.9. Программа вывода треугольника Паскаля

6.3. Оператор *WHILE...WEND*

При разработке программ довольно часто возникает ситуация, когда заранее невозможно установить, сколько раз потребуется повторить цикл. Условие выхода из цикла в этих слу-

чаях определяется ходом решения самой поставленной задачи. Для решения этого вида задач цикл FOR...NEXT не годится. Во всех языках программирования подобные ситуации разрешаются с помощью специальных циклов, каковым и является рассматриваемый оператор цикла WHILE...WEND. Этот оператор цикла способен не только выполнять задачи, для решения которых он и был разработан, но и повторять все задачи цикла FOR...NEXT.

В общем виде рассматриваемый оператор цикла WHILE...WEND выглядит следующим образом:

```
WHILE (условие входа в цикл)
  (тело цикла)
WEND
```

Итак, WHILE...WEND — управляющий оператор, выполняющий блок операторов (составляющих тело цикла) до тех пор, пока выполняется указанное условие. Следовательно, и вход в цикл произойдет лишь в том случае, когда условие цикла истинно. Если же условие цикла не выполняется (т. е. оно ложно), то программа переходит на оператор, следующий за WEND.

Логика цикла WHILE...WEND дана в блок-схеме, приведенной на рис. 6.10.

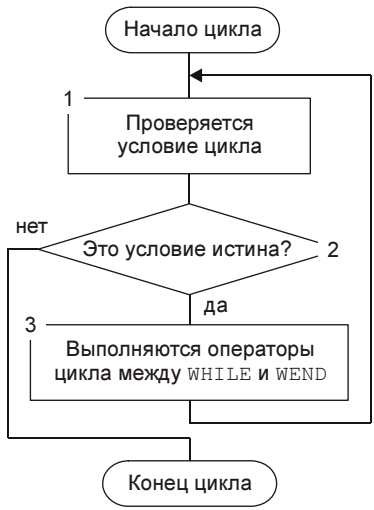


Рис. 6.10. Логика цикла WHILE...WEND

Пример 6.8. Определение высоты подъема

Найти высоту H подъема тела, брошенного под углом t к горизонту с начальной скоростью V, определяемую формулой $H = V^2 \sin^2(t)/(2 \cdot g)$, при $t = 10°,\ 20°,\ 30°, \ldots, 90°$, где $g = 9.81$ м/с^2 — ускорение свободного падения.

Программа 6.12

```
'6W-1cikl.bas
10 CLS        'очистка экрана
20 PRINT
30 PRINT " Высота подъема тела, брошенного"
40 PRINT " под углом к горизонту"
```

```
50 DEFLNG H, T, V
60 PRINT
'==1==== Блок начальных установок ================================
70 t = 10               'начальное значение угла
80 V = 20               'начальное значение скорости м/с
90 g = 9.81             'ускорение свободного падения м/(с*с)
100 PRINT SPC(8); "Угол   Высота"
110 PRINT "       (град)   (м)"
'==2==== Блок расчета и вывода высоты ============================
120 WHILE c <> 1
130 IF t >= 90 THEN c = 1
140 H = V ^ 2 * SIN (t * 3.14 / 180) ^ 2 / (2 * g)
150 PRINT TAB(9); t; TAB(17); H
160 t = t + 10
170 WEND
'================================================================
180 END
```

Операторы `PRINT` в строках 20—40, 60, 100—110 определяют интерфейс программы 6.12, а `PRINT` в строке 150 обеспечивает вывод значений угла и соответствующих им значений высоты. Причем позиционирование вывода в строке 100 производится с помощью функции `SPC`, задающей число пробелов, и пробелами внутри кавычек, в строке 110 — только пробелами внутри кавычек, а в строке 150 — с помощью `TAB`. В строке 50 задается тип переменных, начинающихся с букв `H`, `T`, `V`, как длинные целые, а в строках 70—90 с помощью оператора присваивания вводятся конкретные числовые значения. Цикл `WHILE...WEND` находится в строках 120—170, причем в строках 130—160 — тело цикла. Цикл по условию в строке 120 будет работать до тех пор, пока `c` не станет равно 1. Выход из цикла обеспечивает оператор `IF...THEN` в строке 130, когда угол станет равным 900. Оператор присваивания в строке 160 задает значения угла равными 100, 200, ... , 900. Результат работы программы 6.12 представлен на рис. 6.11.

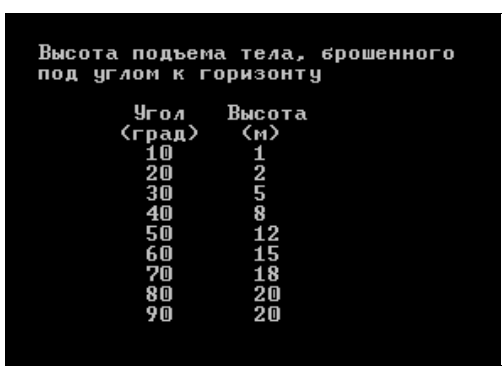


Рис. 6.11. Высота подъема

Пример 6.9. Тест

Разработать программу, выводящую вопрос с вариантами ответа. Оставшееся на раздумье время выводится на экран, как и комментарий выбора варианта ответа.

Программа 6.13

```
'6W-2cikl.bas
10 CLS        'очистка экрана
20 DEFLNG A — Z
30 Tp = 30             'время для размышления (сек.)
40 t1 = TIMER
'==1===== Блок вывода вопроса ============================
50 PRINT "Бейсик сам переделает в PRINT "
60 PRINT
70 PRINT "1 Знак вопроса"
80 PRINT "2 Восклицательный знак"
90 PRINT "3 Решетку"
100 PRINT "4 Знак доллара"
'==2===== Блок выбора ответа =====================================
110 WHILE c <> 1
120 IF t > Tp THEN c = 1: o$ = "Опоздал с ответом                  "
130 t2 = TIMER
140 t = t2 — t1
150 LOCATE 10, 1
160 PRINT "Осталось на размышление "; Tp — t; "сек.   "
170 a$ = INKEY$
180 IF a$ = "1" OR a$ = "2" OR a$ = "3" OR a$ = "4" THEN c = 1: b = 1
190 WEND
'==3===== Блок анализа ===========================================
200 IF b = 1 THEN IF a$ = "1" THEN o$ = "Правильно" ELSE
                                 o$ = "Неправильно"
210 LOCATE 10, 1
220 PRINT "                                                      "
230 LOCATE 10, 1
240 PRINT a$; " "; o$
'=================================================================
250 END
```

В строке 20 все переменные в программе определяются как длинные целые, поэтому секунды на экран выводятся без дробной части. В строке 30 задается определенное для ответа время, а в строке 40 запускается отсчет времени (фиксируется момент начала работы цикла `WHILE...WEND` в строках 110—190). Цикл будет работать до тех пор, пока `c` не станет равным 1, что определяется условием в строке 110. Переменная `t2` в строке 130 постоянно изменяется в соответствии с текущим временем. Операторы в строках 120—140 организуют окончание работы цикла по истечении выделенного на ответ времени (`c` = 1). В строке 160 производится обратный отсчет времени, выводимый на экран посредством оператора `PRINT`, а вывод позиционируется оператором `LOCATE` в строке 150. Функция `INKEY$` в строке 170 фиксирует выбранный вариант ответа, который проверяется оператором `IF...THEN` в строке 180, после чего завершается работа цикла (`c` = 1). В строке 200 оператор `IF...THEN` анализирует выбранный вариант ответа и выдает соответствующий комментарий, выводимый на экран оператором `PRINT` в строке 240. Переключатель `b` фиксирует, что ответ был дан вовремя и исключает появления сообщений `"Правильно"` или `"Неправильно"` в том случае, если время было просрочено. Оператор `PRINT` в строке 220 стирает предыдущее сообщение.

Результат работы программы 6.13 приведен на рис. 6.12.

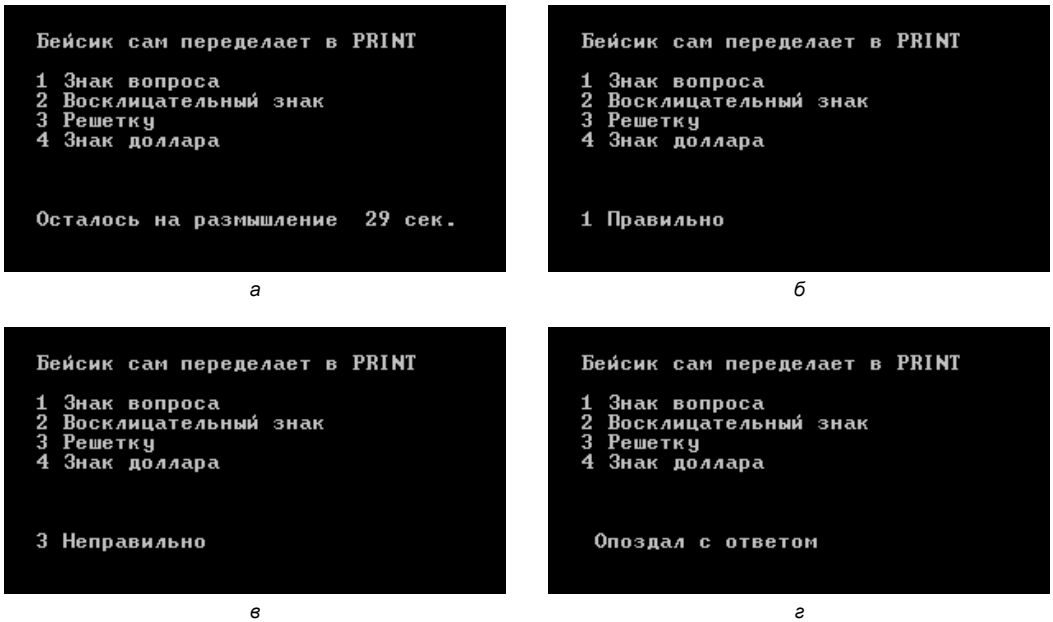


Рис. 6.12. Программы вывода теста: ожидание ответа (*а*); дан правильный ответ (*б*); дан неправильный ответ (*в*); время просрочено (*г*)

Циклы `WHILE...WEND` могут вкладываться друг в друга любое число раз, но каждому оператору `WHILE` должен соответствовать свой `WEND`. Оператор `WHILE` без `WEND` приводит к ошибке "WHILE без WEND" ("WHILE without WEND"), а `WEND` без `WHILE` — к ошибке "WEND без WHILE" ("WEND without WHILE").

В цикл `WHILE...WEND` могут вкладываться и другие циклы, например цикл `FOR...NEXT`, как в следующем примере.

Пример 6.10. Горизонтальное движение шариков

Требуется разработать программу движения с разной скоростью пяти разноцветных шариков.

Программа 6.14

```
'6W-3cikl.bas
10 CLS          'очистка экрана
20 SCREEN 9     'установка экранного режима
'==1===== Внешний цикл ==========================================
30 WHILE x (1) < 640
        '---2----- Вложенный цикл ---------------------------------
40 FOR i = 1 TO 5
50 CIRCLE (x (i), 50 + i * 40), 10, 9 + i
60 PAINT STEP (0, 0), 9 + i, 9 + i
70 x (i) = x (i) + 1 + .1 * i
80 NEXT i
        '-----------------------------------------------------------2---
```

```
90 FOR iv = 1 TO 500: NEXT iv
100 CLS
110 WEND
'=========================================1==================
120 END
```

Промежуточный результат работы программы 6.14 дан на рис. 6.13.

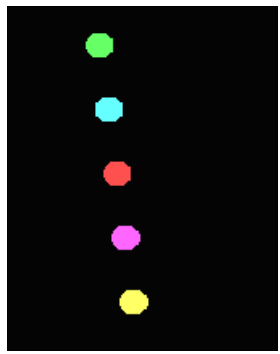


Рис. 6.13. Программа горизонтального движения шариков

Внешний цикл WHILE...WEND в строках 30—110 отвечает за движение шариков и перестает работать по условию в строке 30, когда самый медленный– (верхний) шарик достигнет правой границы экрана. Вложенный цикл FOR...NEXT в строках 40—80 рисует пять шариков и обеспечивает приращение значения координаты по x для каждого из них — строка 70. Вложенный пустой цикл FOR...NEXT в строке 90 задает выдержку времени, после чего изображение стирается оператором CLS в строке 100. Цикл в строках 40—80 снова рисует шарики с новыми значениями координат по x и т. д.

Логические выражения при WHILE по правилам синтаксиса можно и не заключать в круглые скобки, но если они сложные и объединены логическими операторами (AND, OR и пр.), то для большей ясности рекомендуется использовать круглые скобки:

```
WHILE (0 < x) AND (x < 640)
  (тело цикла)
WEND
```

В операторе цикла WHILE...WEND не предусмотрен досрочный выход из цикла, как в операторе FOR...NEXT — EXIT FOR, но подобный досрочный выход достаточно просто организовать самим с помощью условного оператора IF...THEN.

6.4. Оператор *DO...LOOP*

Так же, как и WHILE...WEND, оператор цикла DO...LOOP предназначен для решения задач, в которых невозможно заранее определить число повторений цикла. Однако с его помощью можно реализовать программы, решающие задачи посредством оператора цикла FOR...NEXT.

Пример 6.11. Задача про муху и двух путников

Требуется разработать программу решения задачи о "незабвенной" мухе.

Из двух пунктов, расположенных друг от друга на расстоянии 27 км, вышли навстречу друг другу два путника. Один двигался со скоростью 5 км/час, а другой — со скоростью

4 км/час. Одновременно с одним из путников вылетела муха со скоростью 15 км/час. Долетев до другого путника, муха стукнулась ему в лоб и с той же скоростью полетела обратно. Достигнув лба первого путника, муха изменила направление полета на противоположное. Так она и летала, пока путники не встретились. Сколько километров налетала муха?

Программа 6.15

```
'6D-1cikl.bas
10 CLS        'очистка экрана
20 PRINT " Муха между путниками"
30 PRINT
40 V1 = 5       'скорость 1-го путника
50 V2 = 4       'скорость 2-го путника
60 Vm = 15      'скорость мухи (км / час)
'== Цикл вычисления пути ========================================
70 DO
80 t = t + 1
90 dS = 27 - (V1 + V2) * t
100 Sm = Vm * t
110 LOOP WHILE dS > 0
'================================================================
120 PRINT " Муха налетала"; Sm; "км"
130 END
```

В строках 40—60 с помощью оператора присваивания осуществляется ввод конкретных числовых значений скоростей. В строках 70—110 организован цикл по часам, приращение значения которых реализовано в строке 80. В строке 90 вычисляется расстояние между путниками после каждого часа ходьбы, до тех пор, пока они не встретились, т. е. когда стало `dS` = 0. После этого по условию при `LOOP` в строке 110 цикл завершает работу. В строке 100 также ежечасно вычисляется проделанный мухой путь. Результат после завершения цикла выводится оператором `PRINT` в строке 120.

Результат работы программы 6.15 показан на рис. 6.14.

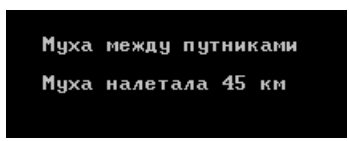


Рис. 6.14. Задача про муху и двух путников

Различаются циклы `DO...LOOP` с предусловием, когда условие цикла, стоящее рядом с ключевым словом `DO`, проверяется в начале цикла, и с постусловием, когда условие цикла, стоящее рядом с ключевым словом `LOOP`, проверяется в конце цикла. Например, в программе 6.15 использован цикл с постусловием. Различия в работе цикла с предусловием и с постусловием иллюстрирует программа 6.16.

Программа 6.16

```
'6D-2Cikl.bas
10 CLS        'очистка экрана
20 PRINT
```

```
30 PRINT " Различие в выполнении цикла "
40 PRINT " DO...LOOP с разными условиями"
50 PRINT
60 INPUT "Задайте значение In = ", In
70 INPUT "Задайте значение Ik = ", Ik
80 PRINT
'==1=== Цикл DO...LOOP с предусловием ============================
90 i = In
100 DO WHILE i <= Ik
110 i = i + 1
120 S1 = S1 + 1
130 LOOP
140 PRINT " Цикл с предусловием  S ="; S1
'==2=== Цикл DO...LOOP с постусловием ============================
150 i = In
160 DO
170 i = i + 1
180 S2 = S2 + 1
190 LOOP WHILE i <= Ik
200 PRINT " Цикл с постусловием  S ="; S2
'================================================================
210 END
```

Результат работы программы 6.16 показан на рис. 6.15.

Как видно из рис. 6.15, при `Ik` > `In` оба варианта цикла работают одинаково, а при `In` > `Ik` цикл с предусловием не выполняется, а цикл с постусловием выполнился один раз. Следовательно, цикл с постусловием выполняется, по крайней мере, один раз.

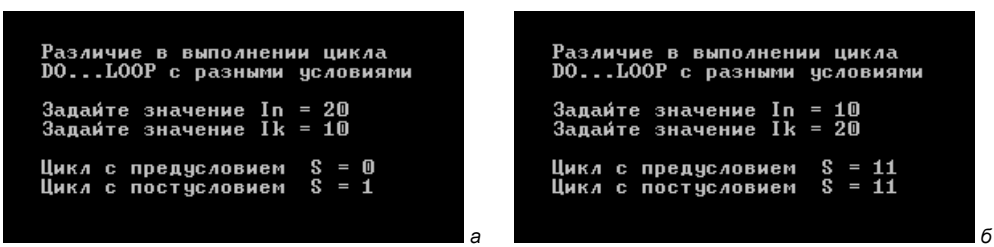


Рис. 6.15. Различие в выполнении цикла с предусловием и с постусловием: начальное значение счетчика цикла `In` больше, чем конечное `Ik` (*а*); начальное значение счетчика цикла `In` меньше, чем конечное `Ik` (*б*)

Кроме того, условия в циклах могут быть с ключевым словом `WHILE` и с ключевым словом `UNTIL`, например:

```
DO WHILE (условие)
  (тело цикла)
LOOP
```

или:

```
DO
  (тело цикла)
LOOP UNTIL (условие)
```

Цикл с ключевым словом WHILE выполняется до тех пор, пока выполняется условие цикла (т. е. пока оно истинно). Причем если условие цикла не выполняется (т. е. оно ложно), то цикл с предусловием вообще работать не будет, а с постусловием выполнится один раз.

Цикл с ключевым словом UNTIL выполняется до тех пор, пока не выполняется условие цикла (т. е. пока оно ложно). Причем если условие цикла выполняется (т. е. оно истинно), то цикл с предусловием вообще работать не будет, а с постусловием выполнится только один раз. Логика цикла DO...LOOP приведена на рис. 6.16.

Если опустить проверку в начале или в конце цикла DO...LOOP, то получится бесконечный цикл. Чтобы выйти из бесконечного цикла DO...LOOP, можно использовать оператор EXIT DO.

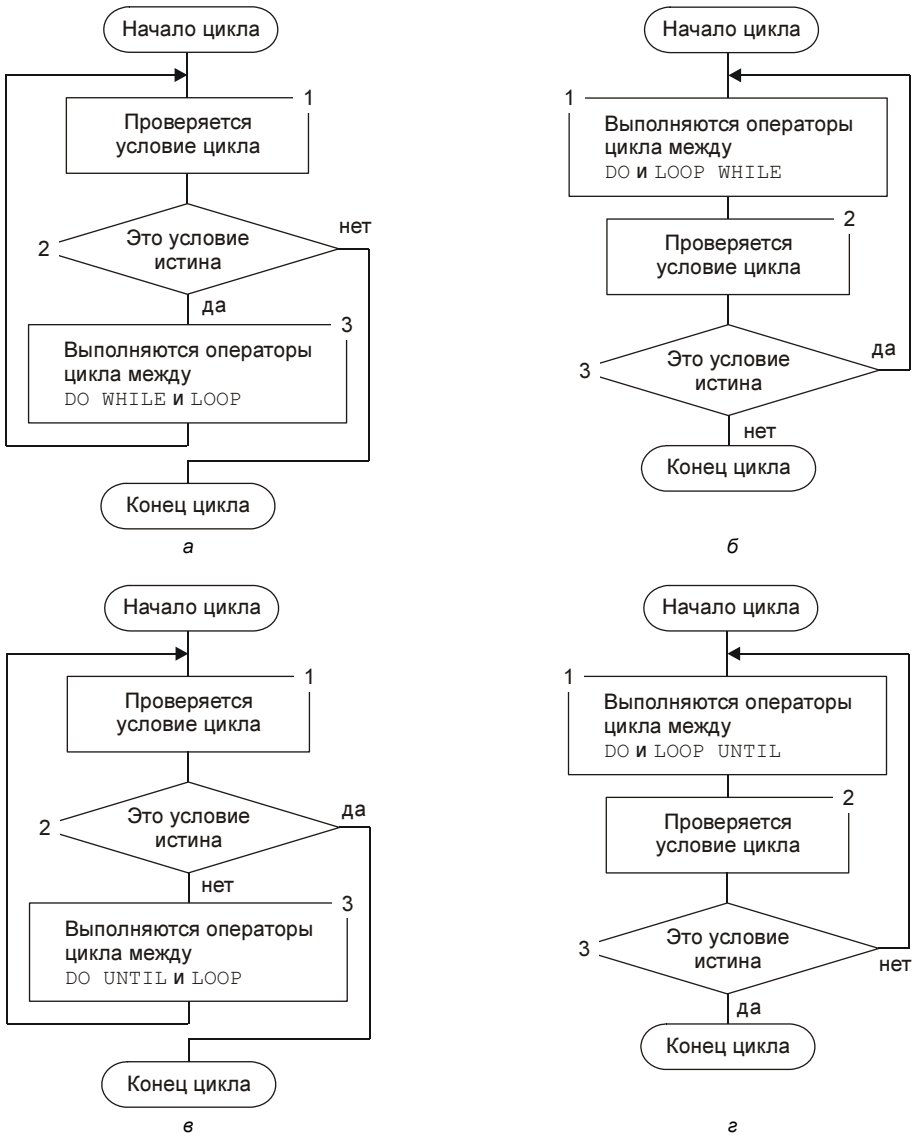


Рис. 6.16. Логика цикла: DO WHILE...LOOP (*а*); DO...LOOP WHILE (*б*); DO UNTIL...LOOP (*в*); DO...LOOP UNTIL (*г*)

Оператор EXIT DO также может оказаться полезным для организации альтернативного выхода из цикла DO...LOOP с проверкой условия в начале или в конце.

Пример 6.12. Финансовая пирамида

Считая известными и равномерными ежемесячные денежные поступления M и возвращаемую ежемесячную процентную ставку β, разработать программу расчета времени "эффективного" существования финансовой пирамиды. Приращение денежных средств рассчитывается по формуле: $\Delta S_t = M - M\beta(t-1)$.

Программа 6.17

```
'6D-3Cikl.bas
10 CLS        'очистка экрана
20 PRINT " Расчет времени ''эффективного'' существования "
30 PRINT "финансовой пирамиды"
40 PRINT
50 INPUT "Ежемесячные поступления в млн. руб. = ", M
60 INPUT "Возвращаемая ежемесячная ставка в % = ", p
'==1=== Блок расчета времени пирамиды ============================
70 DO
80 t = t + 1
90 dS = M - M * p * (t - 1) / 100
100 IF dS <= 0 THEN EXIT DO
110 LOOP
'==2=== Блок вывода результата ===================================
120 PRINT
130 PRINT "Время ''эффективного'' существования ";
140 PRINT "финансовой пирамиды - "; t;
150 SELECT CASE t
      CASE 1: PRINT "месяц"
      CASE 2, 3, 4: PRINT "месяца"
      CASE ELSE: PRINT "месяцев"
160 END SELECT
'==================================================================
170 END
```

Результат работы программы 6.17 дан на рис. 6.17.

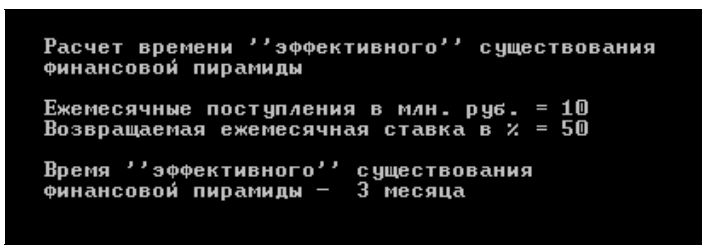


Рис. 6.17. Финансовая пирамида

Ввод исходных данных осуществляется с помощью оператора INPUT в строках 50—60. Цикл для расчета искомого времени "эффективного" существования финансовой пирамиды орга-

низован по месяцам в строках 70—110 (тело цикла — в строках 80—110). Приращение денежных средств dS рассчитывается по формуле в строке 90, а оператор IF...THEN в строке 100, содержащий EXIT DO, обеспечивает завершение работы цикла, когда приращение денежных средств dS перестанет быть положительным. Вывод результата осуществляется посредством операторов в строках 120—160.

С помощью цикла DO...LOOP были решены математическая и экономические задачи. Рассмотрим теперь решение физической задачи.

Пример 6.13. Максимальная дальность полета тела

Дальность полета тела S, брошенного под углом к горизонту t с начальной скоростью V, определяется формулой: $S = V^2 \cdot \sin(2 \cdot t) / g$, где $g = 9.81$ м/с^2 — ускорение свободного падения. Разработать программу, определяющую: при каком значении угла дальность полета тела будет максимальной.

Программа 6.18

```
'6D-4cikl.bas
10 CLS        'очистка экрана
20 DIM S (90)
30 V = 15     'начальная скорость тела (м / с)
'==1==== Блок вычисления дальности ================================
40 DO
50 t = t + 1
60 S(t) = V ^ 2 * SIN (2 * t * 3.14 / 180) / 9.81
70 IF S(t) < S(t - 1) THEN u = 1
80 LOOP WHILE u <> 1
'==2==== Блок вывода результата ===================================
90   PRINT "При начальной скорости"; V; "м / с"
100 PRINT "максимальная дальность полета тела,"
110 PRINT "брошенного под углом к горизонту,"
120 PRINT "будет при угле"; t - 1; "градусов"
130 PRINT "и составляет"; CINT(S(t - 1)); "м"
'====================================================================
140 END
```

Результат работы программы 6.18 приведен на рис. 6.18.

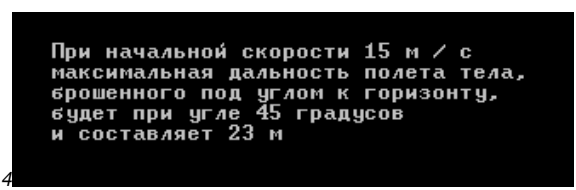


Рис. 6.18. Максимальная дальность полета тела

В строках 40—80 организован цикл по углам t с шагом в 1 градус. В строке 60 вычисляется дальность полета $S(t)$ в зависимости от значения угла. Оператор IF...THEN в строке 70 вы-

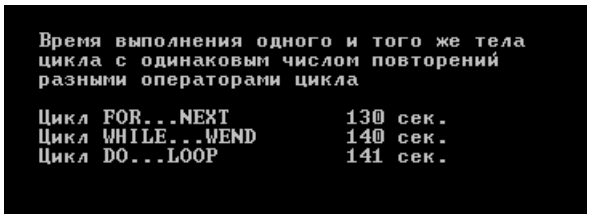
даст команду на окончание цикла на шаге, когда значение $S(t)$ начнет уменьшаться по сравнению с предыдущим.

После того как были рассмотрены три разных оператора цикла, возникает естественный вопрос: а какой же из них при равных условиях работает быстрее? Ответить на этот вопрос поможет программа 6.19.

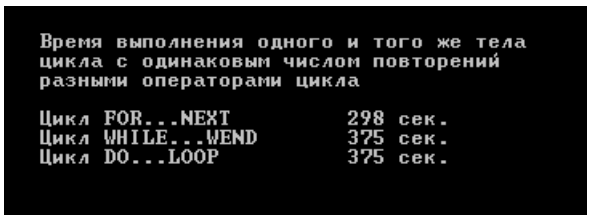
Программа 6.19

```
'6D-5Cikl.bas
10 CLS        'очистка экрана
20 PRINT "Время выполнения одного и того же тела"
30 PRINT "цикла с одинаковым числом повторений"
40 PRINT "разными операторами цикла"
50 PRINT
60 t1 = TIMER
70 n1 = 1000
80 n2 = 20000
'==1==== Цикл FOR...NEXT ========================================
90  FOR i1 = 1 TO n1
100 FOR i2 = 1 TO n2
110 S1 = (S1 + 1) / i2
120 NEXT i2, i1
130 t2 = TIMER
140 PRINT "Цикл FOR...NEXT"; CINT(t2 - t1); "сек."
'==2==== Цикл WHILE...WEND ======================================
150 WHILE i3 <= n1
160 i3 = i3 + 1
170 i4 = 0
180 WHILE i4 <= n2
190 i4 = i4 + 1
200 S2 = (S2 + 1) / i4
210 WEND
220 WEND
230 t3 = TIMER
240 PRINT "Цикл WHILE...WEND"; CINT(t3 - t2); "сек."
'==3==== Цикл DO...LOOP ==========================================
250 DO WHILE i5 <= n1
260 i5 = i5 + 1
270 i6 = 0
280 WHILE i6 <= n2
290 i6 = i6 + 1
300 S3 = (S3 + 1) / i6
310 WEND
320 LOOP
330 t4 = TIMER
340 PRINT "Цикл DO...LOOP"; CINT(t4 - t3); "сек."
'================================================================
350 END
```

Результат работы программы 6.19 показан на рис. 6.19.

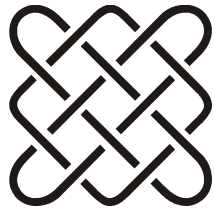


а



б

Рис. 6.19. Сравнение времени работы циклов: с переменными типа Длинные целые (*а*); с переменными типа Обычной точности (*б*)



ГЛАВА 7

Массивы

Эта глава посвящена массивам — одному из эффективных инструментов языков программирования. BASIC — не исключение, и в программах, созданных на этом языке, также широко используются массивы. Так что же это такое и для чего они нужны?

7.1. Назначение массивов

Поясним назначение массивов на конкретном примере. Пусть у нас есть список из 12 студенток, которых нужно разыскать. Но как найти этих "девушек без адреса"? Запишем их адреса в виде табл. 7.1.

Таблица 7.1. *Список студенток*

Номер комнаты	1	2	3	4	5	6
Фамилия	Шевлякова	Федорова	Дулина	Иванова	Васильева	Сидорова
Номер комнаты	7	8	9	10	11	12
Фамилия	Степанова	Тимарина	Сухова	Петрова	Кузнецова	Лескова

Вот теперь они обрели прописку и адрес. Но запомнить целую вереницу значений таблицы в одной простой переменной невозможно. Применять же отдельную переменную для хранения каждой фамилии из таблицы весьма неудобно, даже если их всего 12.

Поэтому во всех языках программирования высокого уровня, а BASIC относится к таким языкам, для хранения подобных данных выделяется целая последовательность ячеек строго одинакового размера, определяемого типом данных, в которой и предполагается размещать эти данные. Количество ячеек устанавливается при описании такой составной переменной. Вся такая последовательность ячеек имеет единое имя и называется *массивом*.

Различают массивы одномерные и многомерные. Одномерные массивы в некоторых литературных источниках также называют рядом, а двумерные — матрицей.

Каждый элемент в массиве имеет номер (индекс), по которому можно обратиться к значению элемента. Индекс — это числовое выражение целого типа.

Размерностью массива называется число индексов, определяющих элемент массива. Например, `x(4)` — значение в одномерном массиве (векторе), `y(1, 2)` — двумерном массиве (матрице).

Размером массива называется количество элементов в массиве. По умолчанию размер любого массива принимается равным 10 элементам.

Массивы могут быть статические и динамические.

Статическими называются массивы, память для размещения которых выделяется интерпретатором еще до начала работы программы, в ходе ее трансляции. Число элементов и размерность таких массивов не могут изменяться в ходе выполнения программы и задаются только при его описании с помощью оператора `DIM`.

Память для динамических массивов отводится лишь тогда, когда в этом возникает необходимость. Убираются они из памяти по ходу выполнения программы по желанию программиста. Очистив память от первого обработанного динамического массива, его место можно последовательно занять любым количеством других динамических массивов.

Если массив не будет объявлен до его использования, то при запуске программы возможно появление сообщения об ошибке "Массив не определен" ("Array not defined").

При задании размеров массива следует учитывать следующие ограничения:

- максимальный размер массива — 65 535 байт;
- максимальное число размерностей — 8;
- максимальный номер индекса — 32 768.

Например, при объявлении числового массива обычной точности, каждый элемент которого занимает 4 байта, максимальный размер массива, указываемый в операторе `DIM`, составляет 65 535/4 = 16 383. При превышении этого значения BASIC выдаст сообщение об ошибке "Индекс вне диапазона" ("Subscript out of range").

На рис. 7.1 поясняется структура одномерного массива.

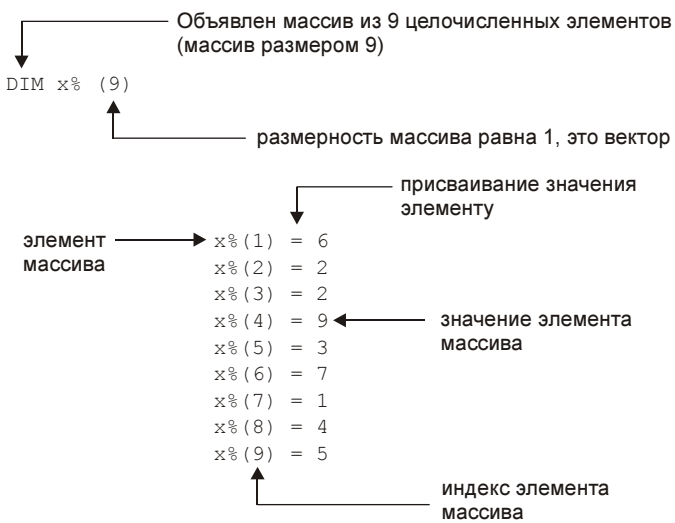


Рис. 7.1. Структура одномерного массива

7.2. Операторы для работы с массивами

Для того чтобы работать с массивами, сначала их надо объявить. Статические массивы объявляются оператором DIM, которым также можно объявлять и переменные других типов. В отличие от оператора DIM, оператор REDIM служит только для объявления динамических массивов. Для обслуживания уже объявленных массивов предназначен оператор ERASE, обнуляющий элементы статических массивов и полностью убирающий из памяти динамические массивы. При необходимости управлять нижней границей массивов можно посредством оператора OPTION BASE.

DIM — оператор, объявляющий один или несколько массивов или переменных и выделяющий для них необходимый объем памяти:

```
DIM x (n1 TO n2, n3 TO n4) AS тип
```

где x — имя массива или переменной;

(n1 TO n2, n3 TO n4) — границы размерности массива — верхняя (n2, n4) и нижняя границы (n1, n3) индексов;

AS *тип* определяет тип массива или переменной: INTEGER, LONG, SINGLE, DOUBLE, STRING.

Ключевое слово TO позволяет указать и нижнюю, и верхнюю границы массива. Следующие операторы эквивалентны (если выполнен оператор OPTION BASE 0 или он отсутствует):

```
DIM x (8, 3)
DIM x (0 TO 8, 0 TO 3)
DIM x (8,0 TO 3)
```

С ключом TO можно указывать не только положительные границы, но и отрицательные. Возможные значения границ находятся в пределах от –32 768 до 32 767, например:

```
DIM x (-8 TO 3)
DIM x (-40 TO -8, -3 TO 3)
```

При обращении к элементу массива, индекс которого лежит вне указанных границ, выдается сообщение об ошибке "Индекс вне диапазона" ("Subscript out of range").

Тип массива возможно задавать посредством суффикса:

```
DIM x$(-8 TO 3)
DIM x%(-40 TO -8, -3 TO 3)
```

Оператор DIM обнуляет все элементы числовых массивов, а элементам символьных массивов присваивает значение пустой строки.

При попытке определить массив оператором DIM после того, как уже было обращение к его элементам, возникает сообщение об ошибке "Массив уже определен" ("Array already dimensioned"). Поэтому оператор DIM лучше всего поставить в самом начале программы.

REDIM — оператор объявления массивов, изменяющий объем памяти, выделенной динамическим массивам, которые обозначаются как $DYNAMIC:

```
REDIM x (n1 TO n2, n3 TO n4) AS тип
```

где x — имя массива;

(n1 TO n2, n3 TO n4) — границы размерности массива — верхняя (n2, n4) и нижняя границы (n1, n3) индексов;

AS *тип* — тип массива.

Ключ TO указывает на то, что определяются и верхняя, и нижняя границы данной размерности массива. Все массивы, описанные данным оператором REDIM, являются динамическими ($DYNAMIC). Оператором REDIM можно лишь изменить границы размерностей массива, но нельзя изменить число размерностей. Приведем правильный пример:

```
DIM x(12, 12) 'объявлен двумерный массив
ERASE x    'очищает массив
REDIM x(15, 18)
```

При его выполнении все пройдет гладко.

А вот другой пример:

```
DIM x(12, 12) 'объявлен двумерный массив
ERASE x    'очищает массив
REDIM x(15, 18, 6)
```

В этом случае появится сообщение об ошибке "Неверное число размерностей" ("Wrong number of dimensions").

OPTION BASE — оператор объявления нижней границы массивов:

```
OPTION BASE n
```

где n равен 0 или 1.

OPTION BASE не обязателен и по умолчанию равен 0. Этот оператор может быть записан в более простом виде и может использоваться только один раз в модуле, до описания всех массивов.

ERASE — оператор управления памятью, обнуляющий элементы статических ($STATIC) массивов или уничтожающий динамические ($DYNAMIC) массивы:

```
ERASE m1, m2, ... , mN
```

где m1, m2, ... , mN — имена описанных массивов.

Оператор ERASE обнуляет числовые элементы статических массивов и присваивает значение пустой строки элементам символьных массивов.

Использование оператора ERASE для динамических массивов освобождает память, занятую этими массивами. Если программа вновь обращается к этим массивам, то они должны быть заново описаны операторами DIM или REDIM. Повторное определение массива оператором DIM без предварительного стирания выдаст сообщение об ошибке "Массив уже определен" ("Array already dimensioned"). Оператор ERASE не требуется, если массивы переопределяются с помощью оператора REDIM.

7.3. Работа с массивами

После того как массив объявлен и сформирован, над ним можно производить разные необходимые операции и преобразования. Наиболее популярными задачами для программирования являются поиск максимального или минимального элемента массива, поиск элемента по заданному условию, сортировки по убыванию или возрастанию или по другому принципу. При обработке двумерных массивов можно отметить такие задачи, как умножение и сложение двух матриц, нахождение следа и транспонирование матрицы, нахождение минора матрицы и пр. Кстати, ввод и вывод матриц также не обойден вниманием. Графическое изображение массивов представлено на рис. 7.2.

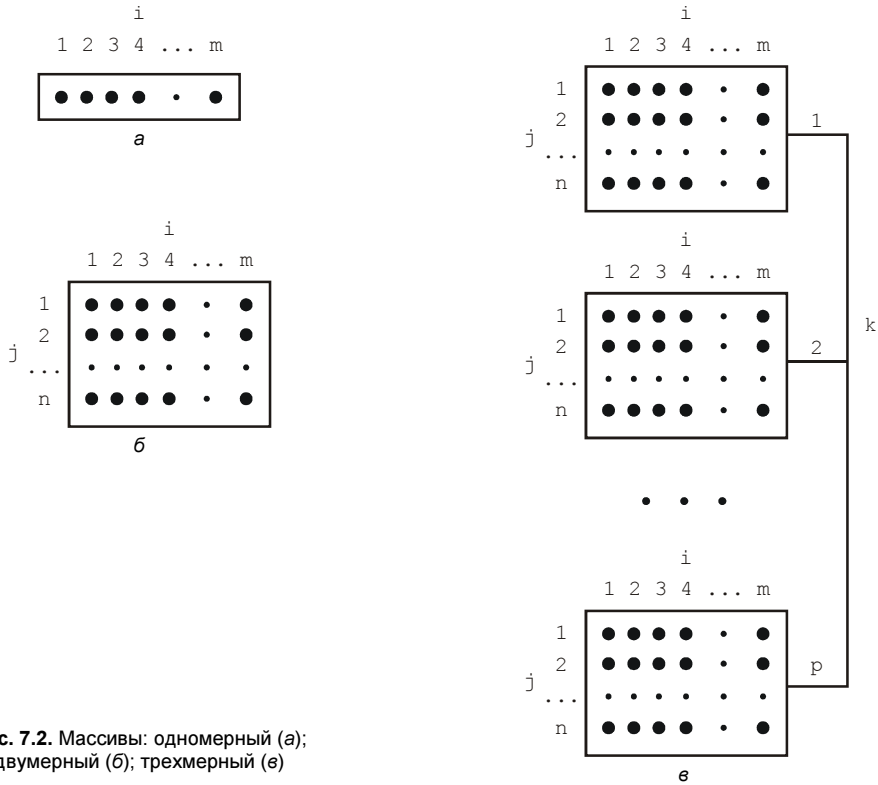


Рис. 7.2. Массивы: одномерный (*а*); двумерный (*б*); трехмерный (*в*)

Для получения практических навыков рассмотрим несколько примеров и начнем с одномерных массивов. Для первого примера вспомним табл. 7.1 и задачу поиска студентки.

Пример 7.1. Поиск в одномерном массиве

Требуется разработать программу поиска студентки.

Программа 7.1

```
'7Cpicok.bas   Список студенток
10 CLS
20 DIM f$(12)
30 DATA Шевлякова,Федорова,Дулина,Иванова,Васильева,Сидорова
40 DATA Степанова,Тимарина,Сухова,Петрова,Кузнецова,Лескова
50 FOR i = 1 TO 12
60 READ f$(i)
70 CLS
80 PRINT "Персона Вашего поиска — "; f$(i)
90 INPUT "Да -1, нет — Enter. Выбор — ", p
100 IF p = 1 THEN
110 PRINT f$(i); " живет в комнате номер"; i
120 GOTO 160
130 END IF
```

```
140 NEXT i
150 PRINT "Других нет"
160 END
```

Результат работы программы 7.1 дан на рис. 7.3.

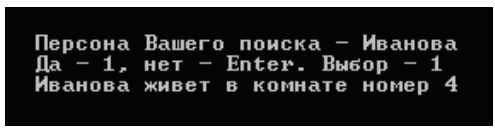


Рис. 7.3. Вывод программы поиска студентки

Одномерный символьный массив объявлен оператором DIM в строке 20. Для ввода и обработки массива создан цикл с оператором FOR...NEXT в строках 50—140. Ввод элементов массива производится посредством операторов READ и DATA в строках 30—40, 60. Оператор INPUT в строке 90 реализует выбор, который обрабатывается с помощью условного оператора в блочной форме IF...THEN...END IF в строках 100—130.

Следующий пример также иллюстрирует использование одномерных массивов.

Пример 7.2. Псевдографическая цветомузыка

Требуется разработать программу, реализующую псевдографический варианта цветомузыки.

Программа 7.2

```
'7CvetMuz.bas    Цветомузыка (псевдографический вариант)
10 CLS
20 RANDOMIZE 32767 - TIMER
30 DIM x(80), b$(80)
'==1===== Блок ввода элементов массивов ============================
40 DATA "К",12,"К",4,"К",12,"З",10,"С",3,"З",10,"Ж",14,"Ж",14
50 DATA "Г",11,"Г",11,"Ф",13,"С",5,"Ф",13,"С",9,"Ф",1,"С",9
60 FOR i = 1 TO 80 STEP 5
70 READ b$(i), x(i)
80 b$(i + 1) = b$(i)
90 b$(i + 2) = b$(i)
100 b$(i + 3) = b$(i)
110 b$(i + 4) = b$(i)
120 x(i + 1) = x(i)
130 x(i + 2) = x(i)
140 x(i + 3) = x(i)
150 x(i + 4) = x(i)
160 NEXT i
'==2===== Блок игры цвета ==========================================
170 DO
180 GOSUB 300                      'рисует поле в первоначальном виде
190 FOR iv = 1 TO 10000: NEXT iv       'выдержка времени
200 IF INKEY$ = CHR$(27) THEN EXIT DO  'выход из цикла (клавиша <Esc>)
210 GOSUB 500                          'стирает семь колонок
```

```
220 FOR iv = 1 TO 10000: NEXT iv       'выдержка времени
230 LOOP
'=================================================================
240 END

'==3===== Подпрограмма вывода цветового поля ======================
300 FOR j = 1 TO 80
310 COLOR x(j)
320 c$ = b$(j)
330 FOR i = 1 TO 21
340 LOCATE 2 + i, 0 + j: PRINT c$
350 NEXT i, j
      '----- Формирование верхушек поля ----------------------------
360 FOR mx = 1 TO 80
370 d = (mx + 5) MOD 5                 'формирование по пятеркам
380 IF d = 1 OR d = 0 THEN n = 4
390 IF d = 2 OR d = 4 THEN n = 3
400 IF d = 3 THEN n = 2
      '----- Стирает вертикальный столбик --------------------------
410 FOR i = 1 TO n
420 LOCATE 0 + i, mx: PRINT " "        'стирает пробелом
430 NEXT i
      '-----------------------------------------------------------
440 NEXT mx
450 RETURN
'==4===== Подпрограмма стирания семи колонок ======================
500 xm1 = INT(1 + 60 * RND(1))
510 FOR j = 1 TO 7
520 xm = xm1 + 3 * (j - 1)
530 FOR i = 1 TO 20
540 LOCATE 0 + i, xm: PRINT " "
550 NEXT i, j
560 RETURN
'=================================================================
```

Результат работы программы 7.2 приведен на рис. 7.4.

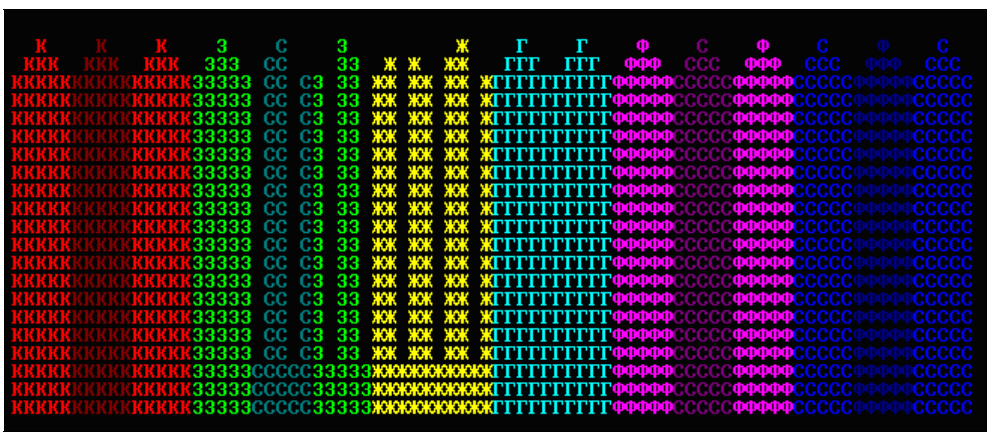


Рис. 7.4. Программа цветомузыки

Впрочем, цветомузыка — название достаточно условное. Скорее, это игра цветом, поскольку музыка здесь ни при чем.

Итак, выводится цветное поле, состоящее из 80 колонок (по числу позиций по горизонтали), причем каждая колонка поля имеет свой цвет. Колонка красного цвета состоит из красных букв "К", колонка зеленого цвета состоит из зеленых букв "З", желтого — из желтых букв "Ж", голубого — из голубых букв "Г", синего — из синих букв "С", фиолетового — из фиолетовых "Ф". В свою очередь, все поле состоит из 16 блоков по 5 колонок одинакового цвета в блоке. Вершина каждого блока оформляется, как показано на рис. 7.4.

Игра цветом заключается в следующем: "гребенкой" из 7 колонок стирается часть поля (GOSUB 500 в строке 210), а затем после выдержки времени восстанавливает исходный вид поля (GOSUB 300 в строке 180). Это повторяется до выхода из программы (оператор EXIT DO в условном операторе IF...THEN в строке 200). Особенностью программы является то, что последовательность номеров выводимых цветов не поддается какой-либо закономерности. Поэтому, чтобы их упорядочить с целью применения циклов, и надо использовать массив.

Такой одномерный массив x(80) задается оператором DIM в строке 30 программы 7.2. Второй массив b$(80) в той же строке предназначен для хранения букв, составляющих колонки. Для ввода организован цикл, и посредством оператора цикла FOR...NEXT в строках 60—160, в теле которого находится оператор READ в строке 70, читающий данные из операторов DATA в строках 40—50, и осуществляется этот ввод. Поскольку каждый блок поля состоит из пяти колонок, то STEP в операторе цикла в строке 60 равняется 5. А пятерку для ввода букв вместе с оператором составляют операторы присваивания в строках 80—110 и для номеров цвета в строках 120—150.

Теперь рассмотрим подпрограмму вывода цветового поля. Для этого служит двойной цикл с оператором FOR...NEXT в строках 300—350. Причем вложенный цикл с оператором FOR...NEXT в строках 330—350 производит вывод одной колонки, а внешний цикл задает позицию вывода, цвет (строка 310) и букву (строка 320). Формирование верхушек "пятерочных" блоков производится путем стирания лишних знаков вертикальным столбиком из пробелов, выводимым оператором цикла FOR...NEXT в строках 410—430, а размеры этого столбика определяют условные операторы IF...THEN в строках 380—400.

В подпрограмме вывода стирающей "гребенки" из 7 вертикальных столбиков из пробелов формирование столбика обеспечивает оператор цикла FOR...NEXT в строках 530—550, а начальная позиция вывода "гребенки" задается случайным образом в строке 500.

Следующие два примера посвящены работе с двумерными массивами или матрицами. Подобные примеры особенно популярны в литературе по программированию при рассмотрении циклов. Вывод матриц уже рассматривался в *главе 4*. Напомним, что под $(m \times n)$-матрицей понимается прямоугольная таблица:

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{vmatrix},$$

состоящая из m строк и n столбцов.

Пример 7.3. Обнуление элементов матрицы

Требуется разработать программу ввода, обнуления элементов матрицы по заданному условию и вывода результата. Обнуляются элементы меньше 4.

Программа 7.3

```
'7Matric1.bas    Обнуление элементов матрицы
10 CLS
20 RANDOMIZE 32767 — TIMER
30 DIM x(8, 8)
40 PRINT SPC(4); "Исходная матрица"
50 PRINT
60 FOR k = 1 TO 3
70 FOR j = 1 TO 8
80 FOR i = 1 TO 8
90 SELECT CASE k
     CASE 1: x(j, i) = INT(1 + 9 * RND(1))
                    PRINT x(j, i);
      CASE 2: IF x(j, i) <= 3 THEN x(j, i) = 0
                      IF j = 8 THEN
                      LOCATE 12, 1
                      PRINT "  Обработанная матрица"
                      END IF
       CASE 3: PRINT x(j, i);
100 END SELECT
110 NEXT i
120 PRINT
130 NEXT j, k
140 END
```

Результат работы программы 7.3 показан на рис. 7.5.

Рис. 7.5. Программа обнуления элементов матрицы

Если писать программу "в лоб", то она будет состоять из трех циклов: цикл для ввода элементов матрицы, цикл для обнуления и цикл для вывода получившейся матрицы, причем каждый из циклов является двойным или вложенным. Однако, используя условный оператор SELECT CASE и добавив еще один внешний цикл FOR...NEXT с числом повторений 3 (строки 60—130), можно свести все к одному двойному циклу FOR...NEXT в строках 70—130. Матрица x(8, 8) задается оператором DIM в строке 30. Ее элементы в диапазоне от 1 до 9 определяются случайным образом в CASE 1. Следующий CASE 2 обеспечивает обнуление элементов, меньших 4, а CASE 3 выводит результирующую матрицу.

Итак, мы установили, что двумерный массив предназначен для запоминания элементов прямоугольной таблицы, каковой и является платежная ведомость какого-нибудь структурного подразделения.

Пример 7.4. Вывод платежной ведомости

Требуется разработать программу вывода платежной ведомости.

Программа 7.4

```
'7Vedomct.bas    Ведомость
10 CLS
20 DIM x(7, 3), f$(7)
30 PRINT SPC(6); "Платежная ведомость"
40 PRINT " № Фамилия   Оклад Премия Итого"
'==1===== Блок ввода фамилий ======================================
50 FOR i = 1 TO 7
60 DATA Алов,Борисов,Ветров,Куров,Наумов,Петинов,Петров
70 READ f$(i)
80 NEXT i
'==2===== Блок ввода и вывода элементов матрицы ===================
100 FOR j = 1 TO 7
110 PRINT j; f$(j); TAB(14);
120 FOR i = 1 TO 3
130 DATA 5000,1500,6500, 6000,1800,7800, 6000,1800,7800, 4000,1200,5200
140 DATA 5000,1500,6500, 7000,2000,9000, 4000,1200,5200
150 READ x(j, i)
160 PRINT x(j, i);
170 NEXT i
180 PRINT
190 NEXT j
'===================================================================
200 END
```

Результат работы программы 7.4 приведен на рис. 7.6.

В строке 20 программы 7.4. оператор DIM задает числовой двумерный массив x(7, 3) для значений платежной ведомости и одномерный символьный массив f$(7) для фамилий сотрудников. Операторы PRINT в строках 30—40 формируют шапку ведомости, а оператор PRINT в строке 110 — первые два столбца ведомости. Ввод фамилий и элементов ведомости осуществляется аналогично — посредством операторов READ/DATA, поставленных в цикле FOR...NEXT, только для элементов ведомости и, естественно, с помощью двойного или вложенного цикла. Вывод элементов ведомости осуществляется оператором PRINT, находящемся в том же двойном цикле с оператором FOR...NEXT в строках 100—190.

Рис. 7.6. Программа вывода платежной ведомости

Применение трехмерных массивов чаще всего требуют задачи, в которых имеется таблица, значения элементов которой меняются во времени. Например, таблица температуры воздуха по месяцам. В книге [8] приводится пример изменения успеваемости учащихся по учебным четвертям. В нашем следующем примере результаты в таблице изменяются по годам.

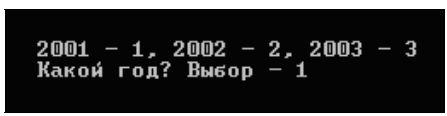
Пример 7.5. Вывод трехмерной матрицы

Разработать программу вывода результатов игр за три года четырех футбольных команд.

Программа 7.5

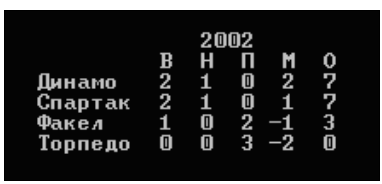
```
'7futbol.bas   Таблицы игр команд
10 CLS
20 DIM f(3, 4, 5), g(3), g1$(4), g2$(4), g3$(4)
'==1===== Блок ввода данных ========================================
30 g(1) = 2001: g(2) = 2002: g(3) = 2003                    'года
    '----- Ввод названий команд ----------------------------------
40 FOR i = 1 TO 4
50 READ g1$(i), g2$(i), g3$(i)
60 DATA Спартак,Динамо,Спартак,Динамо,Спартак,Торпедо
70 DATA Торпедо,Факел,Динамо,Факел,Торпедо,Факел
80 NEXT i
    '----- Ввод результатов игр ----------------------------------
100 DATA 3,0,0,6,9, 2,0,1,1,6, 0,1,2,-3,1, 0,1,2,-4,1
110 DATA 2,1,0,2,7, 2,1,0,1,7, 1,0,2,-1,3, 0,0,3,-2,0
120 DATA 2,1,0,4,7, 2,0,1,1,6, 1,1,1,-2,4, 0,0,3,-3,0
130 kn = 1: Kk = 3: GOSUB 300
'==2===== Блок выбора года ==========================================
140 e = 1
150 PRINT "2001 — 1, 2002 — 2, 2003 — 3"
160 INPUT "Какой год? Выбор — ", f
170 kn = f: Kk = f: GOSUB 300
'====================================================================
200 END

'==3===== Подпрограмма работы с матрицей ===========================
300 CLS
305 FOR k = kn TO Kk
310 IF e = 1 THEN  ELSE 325
315 PRINT SPC(11); g(f)
320 PRINT SPC(9); "В  Н  П  М  О"
```

```
325 FOR j = 1 TO 4
   '----- Вывод названий команд -----------------------------------
330 IF e = 1 THEN  ELSE 340
335 IF f = 1 THEN PRINT g1$(j); TAB(9);
340 IF f = 2 THEN PRINT g2$(j); TAB(9);
345 IF f = 3 THEN PRINT g3$(j); TAB(9);
   '----- Ввод и вывод элементов массива --------------------------
350 FOR i = 1 TO 5
355 IF e = 0 THEN READ f(k, j, i)
360 IF e = 1 THEN PRINT f(k, j, i);
365 NEXT i
370 PRINT
375 NEXT j
380 PRINT
385 NEXT k
390 RETURN
'=================================================================
```



а

б



в

г

Рис. 7.7. Программа вывода трехмерной матрицы: выбор года (*а*); результаты игр за разные года (*б*, *в*, *г*)

В строке 20 программы 7.5 оператор `DIM` объявляет 5 массивов: трехмерный числовой `f(3, 4, 5)` для результатов игр, числовой одномерный `g(3)` для запоминания сезонов и три символьных одномерных `g1$(4)`, `g2$(4)`, `g3$(4)` для названий команд. Запоминание названий команд требует трех массивов, поскольку в разных сезонах они занимали разные места.

Ввод данных осуществляется двумя способами. Из-за небольшого объема сезоны (года) вводятся не в цикле, а посредством трех операторов присваивания в строке 30. Названия команд и значения элементов трехмерного массива записываются в память с помощью операторов `READ` и `DATA`, поставленных в операторы цикла `FOR...NEXT` в строках 40—80 и 305—385 соответственно. Причем последний цикл удобнее оформить в виде подпрограммы, которая производит также и вывод таблиц. Отметим, что число повторений цикла в операторе `FOR...NEXT` в строках 305—385 при вводе и выводе данных разное, при вводе — 3 (задано в строке 130), а при выводе — 1 (строки 160—170).

Теперь о подпрограмме работы с матрицей — строки 300—390. Операторы `PRINT` в строках 315—320 выводят шапку таблицы, причем условный оператор `IF...THEN` в строке 310 блокирует вывод шапки при вводе данных в трехмерный массив. Для читателей, далеких от

спорта, дадим расшифровку сокращений в шапке: "В" — выигрыши, "Н" — ничьи, "П" — поражения, "М" — мячи (разница мячей) и "О" — очки. Три условных оператора IF...THEN в строках 335—345 выводят первые два столбца таблицы, состоящие из названий команд, причем порядок следования команд зависит от выбранного сезона (определяет значение переменной f, вводимой оператором INPUT в строке 160). Условный оператор в строке 330 блокирует вывод боковика при вводе данных. И, наконец, условные операторы IF...THEN в строках 355—360 разделяют действия для ввода и вывода данных при совмещении двух циклов в циклическом операторе FOR...NEXT в строках 305—385.

Значения элементов трехмерного массива могут изменяться не только в зависимости от времени, но и по другим параметрам. Это подтверждается следующим примером 7.6.

Пример 7.6. Вывод баллов за конкурсы КВН

Четыре команды КВН соревнуются в трех конкурсах, оцениваемых пятью членами жюри. Максимальный балл в конкурсе "Приветствие" — 5, в конкурсе "Разминка" — 4 и конкурсе "Музыкальный" — 7 баллов.

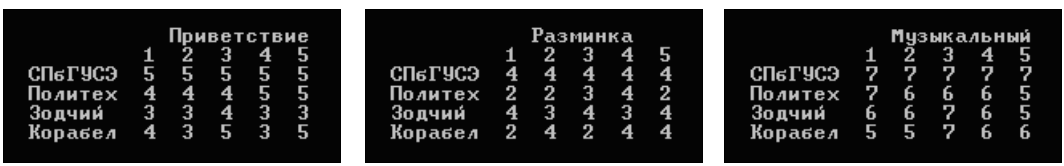
Требуется разработать программу вывода таблиц оценок жюри командам КВН за три конкурса.

Программа 7.6

```
'7KVN.bas   Оценки команд КВН за три конкурса
10 CLS
20 DIM f(3, 5, 5), z$(3), k$(5)
'==1===== Блок ввода данных ========================================
30 z$(1) = "Приветствие"
40 z$(2) = "Разминка"
50 z$(3) = "Музыкальный"
     '----- Ввод названий команд ----------------------------------
60 FOR i = 1 TO 5
70 READ k$(i)
80 DATA ,СПбГУСЭ,Политех,Зодчий,Корабел
90 NEXT i
    '----- Ввод оценок -------------------------------------------
100 DATA 1,2,3,4,5, 5,5,5,5,5, 4,4,4,5,5, 3,3,4,3,3, 4,3,5,3,5
110 DATA 1,2,3,4,5, 4,4,4,4,4, 2,2,3,4,2, 4,3,4,3,4, 2,4,2,4,4
120 DATA 1,2,3,4,5, 7,7,7,7,7, 7,6,6,6,5, 6,6,7,6,5, 5,5,7,6,6
130 kn = 1: Kk = 3: GOSUB 300
'==2===== Блок выбора конкурса ======================================
140 a = 1
150 PRINT "Приветствие — 1"
160 PRINT "Разминка — 2"
170 PRINT "Музыкальный — 3"
180 INPUT "Какой конкурс? Выбор — ", v
190 kn = v: Kk = v: GOSUB 300
'====================================================================
200 END

'==3===== Подпрограмма работы с матрицей ============================
300 CLS
310 FOR k = kn TO Kk
```

```
320 IF a = 1 THEN  ELSE 340
330 PRINT SPC(11); z$(v)
340 FOR j = 1 TO 5
   '----- Вывод названий команд -----------------------------------
350 IF a = 1 THEN PRINT k$(j); TAB(9);
   '----- Ввод и вывод элементов массива --------------------------
360 FOR i = 1 TO 5
370 IF a = 0 THEN READ f(k, j, i)
380 IF a = 1 THEN PRINT f(k, j, i);
390 NEXT i
400 PRINT
410 NEXT j
420 PRINT
430 NEXT k
440 RETURN
'================================================================
```

Результат работы программы 7.6. дан на рис. 7.8.



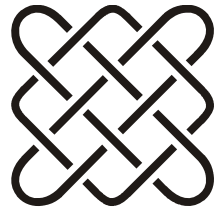
а *б* *в*

Рис. 7.8. Программа вывода баллов за три конкурса КВН

В строке 20 программы 7.6 оператором `DIM` объявляется трехмерный числовой массив `f(3, 5, 5)` для оценок жюри и два одномерных символьных массива: `z$(3)` — для названий конкурсов, `k$(5)` — для названий команд КВН. Ввод и вывод данных такие же, как в примере 7.5. Вложенный цикл для работы с трехмерным массивом также оформлен подпрограммой.

Предлагаем читателю самостоятельно доработать программу 7.6 таким образом, чтобы она выводила средний балл за конкурс, а также суммарный балл за три конкурса.

Надеемся, что приведенные примеры убедили вас в эффективности использования массивов.



ГЛАВА 8

Работа с графикой

8.1. Графические операторы

Начинать программы, строящие графические изображения, рекомендуется с оператора `CLS` — оператора, очищающего экран дисплея.

Оператор `CLS` может иметь следующие аргументы:

- `CLS 0` — очищает и текстовый и графический экраны;
- `CLS 1` — очищает только графический экран, если он активен;
- `CLS 2` — очищает только текстовый экран, исключая нижнюю строку.

Если все аргументы этого оператора опущены, то очищаются и графический, и текстовый экраны, установленные оператором `VIEW` или взятые по умолчанию.

Пример 8.1. Действия оператора *CLS*

На графическом экране рисуются линии. На текстовом экране выводятся сообщения. Графический и текстовый экраны очищаются раздельно.

Программа 8.1

```
'8CLS.bas
10 CLS
20 SCREEN 9
30 RANDOMIZE 32767 - TIMER
40 LOCATE 20, 2
50 PRINT "Для выхода нажмите любую клавишу"
'==1===== Блок установки экранов вывода ============================
60 VIEW (10, 10)-(200, 120), , 11   'графический
70 VIEW PRINT 10 TO 15              'текстовый
'==2===== Блок рисования и вывода сообщения ========================
80 a = 1: b = 1
90 DO
100 k = k + 1
110 d = INT(1 + 7 * RND(1))
120 c = 8 + d
```

```
    '----------- Рисование линий ----------------------------------
130 IF k MOD 70 = 0 THEN a = -a: CLS 1
140 SELECT CASE a
CASE -1: x = 100: y = 80
CASE 1:  x = 20: y = 20
150 END SELECT
160 LINE (x, y)-STEP(30 + 4 * d * a, 20 * a), c
    '------------ Вывод сообщений ---------------------------------
170 IF k MOD 90 = 0 THEN b = -b: CLS 2
180 SELECT CASE b
CASE 1:   PRINT "Никого нет лучше нашего Хасана"
CASE -1:  PRINT "Никого нет лучше вашего Меня"
190 END SELECT
200 LOOP UNTIL INKEY$ <> ""
'=========================================================
210 END
```

Результат работы программы 8.1 показан на рис. 8.1.

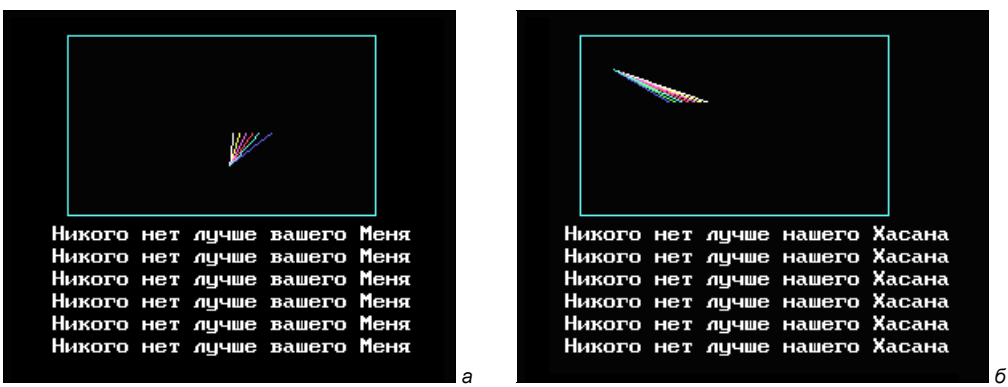


Рис. 8.1. Иллюстрация действия оператора CLS

После установки экранов вывода в строках 60—70 и строках 90—200 организуется цикл DO...LOOP для рисования цветных линий в графическом экране (строки 130—160) и вывода сообщений в текстовом (строки 170—190). Конец и цвет линий (c в строке 120) задается случайным образом операторами RND в строке 110 и оператором LINE в строке 160 (в скобке после STEP). Отметим, что для работы RND требуется выполнить оператор RANDOMIZE 32767 – TIMER в строке 30. Через каждые 70 шагов цикла графический экран очищается посредством оператора CLS 1 в IF...THEN в строке 130, а также меняются координаты начальной точки линий с помощью переключателя a. Для изменения выводимого сообщения через каждые 90 шагов цикла используется переключатель b в операторе IF...THEN в строке 170. А оператор CLS 2 в той же строке с той же периодичностью очищает тестовый экран. Экранный режим определяет оператор SCREEN в строке 20.

SCREEN — это графический оператор, устанавливающий спецификацию экрана. Целое число после оператора, например SCREEN 9, указывает на режим экрана. В табл. 8.1 приведены основные характеристики режимов.

Таблица 8.1. *Экранные режимы для адаптеров VGA и SVGA*

Номер режима	Разрешение	Количество цветов	Количество страниц видеопамяти
0	Текстовый режим	16	1—8
1	320×200	4	1
2	640×200	2	1
7	320×200	16	8
8	640×200	16	4
9	640×350	16	2
10	640×350	2	2
11	640×480	2	1
12	640×480	16	1
13	320×200	256	1

Примечание

Для адаптера VGA (Video Graphics Array) максимальное разрешение — 640×480, и количество цветов — 256. Для адаптера SVGA (Super Video Graphics Array) разрешение — до 1024×1024, количество цветов — до 16 миллионов.

Для работы обычно выбирается один "любимый" режим с тем, чтобы по мере его освоения помнить наизусть его разрешение. Кроме того, результаты действия графических операторов будут ожидаемыми и предсказуемыми. Авторы рекомендуют использовать `SCREEN 9`. Экранные цвета задаются с помощью оператора `COLOR`:

```
COLOR n1, n2
```

Этот оператор предназначен для изменения цвета символов или цвета точек и линий, если цвет в соответствующих операторах (например, `PSET`, `LINE`, `CIRCLE`) не указан в явном виде, на `n1` (основной цвет, цвет переднего плана, цвет пера) и цвета экрана — на `n2` (цвет фона, цвет заднего плана).

Например, следующий оператор устанавливает темно-синий основной цвет и цвет фона — белый:

```
COLOR 1, 15
```

Чтобы изменить только цвет символов (основной), оставляя прежним цвет фона, нужно выполнить оператор:

```
COLOR 14
```

Для изменения только цвета фона, оставляя прежним цвет символов (основной), используйте оператор:

```
COLOR , 15
```

Возможные цвета указаны в табл. 8.2.

Таблица 8.2. *Возможные цвета*

Номер	Цвет	Номер	Цвет
0	Черный	8	Темно-серый
1	Синий	9	Светло-синий
2	Зеленый	10	Светло-зеленый
3	Голубой	11	Светло-голубой
4	Красный	12	Светло-красный
5	Фиолетовый	13	Васильковый
6	Коричневый	14	Желтый
7	Светло-серый	15	Ярко-белый

Пример 8.2. Вывод символов разными цветами

Следующая программа выводит символы разного цвета.

Программа 8.2

```
'8COLOR.bas
10 CLS
20 SCREEN 9
30 FOR i = 0 TO 15
40 COLOR i
50 PRINT i; SPC(1);
60 NEXT i
70 END
```

Результат работы программы 8.2 отображен на рис. 8.2.

Рис. 8.2. Вывод символов разным цветом

Теперь рассмотрим графические операторы, рисующие так называемые графические примитивы (операторы `PSET`, `PRESET`, `LINE`, `CIRCLE`).

`CIRCLE` — графический оператор, рисующий эллипс или окружность с определенным центром и радиусом:

```
CIRCLE (x, y), R, cv, nd, kd, sj
```

где `(x, y)` — экранные координаты центра окружности или эллипса;

`R` — радиус окружности или эллипса в текущей координатной системе;

`cv` — атрибут цвета, по умолчанию — основной цвет;

`nd` — начало дуги;

`kd` — конец дуги;

`sj` — коэффициент сжатия. Коэффициент сжатия — это отношение радиуса по `y` к радиусу по `x`. По умолчанию этот коэффициент принимается таким, как для рисования окружности. Отношение вычисляется следующим образом:

4 * (*y*-координата точки / *x*-координата точки) / 3,

где *y*-координаты точки и *x*-координаты точки определяются разрешением экрана. Например, если оператором `SCREEN 1` установлен режим экрана 1, то разрешение экрана будет 320×200. Следовательно, коэффициент сжатия будет равен:

4 * (200 / 320) / 3 = 5/6.

Если коэффициент сжатия < 1, то будет сжатие радиуса по `x`, а если коэффициент сжатия > 1, то будет сжатие радиуса по `y`.

Параметры `nd` и `kd` используются для рисования дуг. Их значения находятся в пределах от `-2 * PI` до `2 * PI` радиан, где `PI` = 3.14. По умолчанию `nd` = 0, а `kd = 2 * PI`. Если `nd` и `kd` > 0, то оператор `CIRCLE` рисует только дугу, а если `nd` и `kd` < 0, то оператор `CIRCLE` рисует дугу и радиус до этой точки. Начальный угол должен быть меньше конечного. Если указан конец дуги без начала, то дуга рисуется от указанного конца до 0. Нарисовать радиус при `nd` = 0 невозможно. В этом случае для рисования радиуса (горизонтального вектора) используется очень маленькое отрицательное значение, например –0.001. Чтобы нарисовать сектор в четверть круга с координатами `x` = 100, `y` = 100 и радиусом `R` = 50, используется оператор:

```
CIRCLE (100, 100), 50, 1, -0.001, -1.57
```

Используя тригонометрическую окружность, представленную на рис. 8.3, удобно определять начало и конец дуг, рисуемых оператором `CIRCLE`. Так, например, как видно из рисунка, верхнюю полуокружность с радиусами (диаметром) нарисует:

```
CIRCLE (x, y), R, cv, -0.01, -3.14
```

А нижнюю полуокружность без радиусов:

```
CIRCLE (x, y), R, cv, 3.14
```

(поскольку параметр, определяющий конец дуги опущен, то по умолчанию дуга рисуется до 6.28, т. е. до конца).

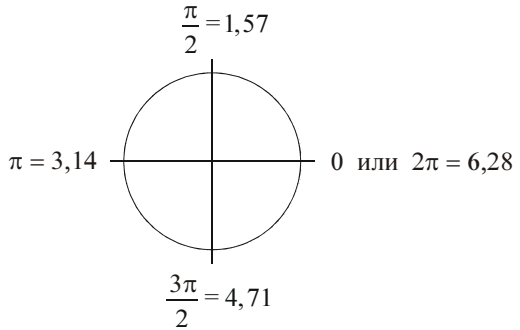


Рис. 8.3. Тригонометрическая окружность

Можно опускать аргументы в середине оператора, но следует обязательно ставить запятые. Например, в следующем операторе пропущен цвет:

```
CIRCLE (100, 100), R, , 0, -1.57
```

Если опущены последние аргументы, то запятые не ставятся.

Последняя точка рисования после завершения оператора — центр окружности. Координаты вне центра окружности могут быть за пределами экрана. Координаты центра окружности можно указывать относительно последней точки рисования, предшествующей оператору CIRCLE. Для этого используется STEP:

```
CIRCLE STEP(100, 100), R
```

Если координаты последней точки рисования были x = 50 и y = 20, то приведенный оператор CIRCLE нарисует окружность с центром в точке x = 150 и y = 120.

Пример 8.3. Использование оператора *CIRCLE*

Следующая программа демонстрирует возможности оператора CIRCLE.

Программа 8.3

```
'8CIRCLE.bas
10 CLS
20 SCREEN 9
30 DATA 100,70,0,6.28,0.8,10,12,"а)"
40 DATA 220,70,0,3.14,0.8,10,28,"б)"
50 DATA 340,70,-0.01,-4.71,0.8,10,44,"в)"
60 DATA 460,70,0,6.28,0.2,10,60,"г)"
70 DATA 100,200,0,6.28,1.5,20,12,"д)"
80 DATA 220,200,3.14,6.28,0.3,20,28,"е)"
90 DATA 340,200,-1,-6,1.5,20,44,"ж)"
100 FOR i = 1 TO 7
110 READ x, y, nd, kd, sj, my, mx, m$
120 CIRCLE (x, y), 50, 15, nd, kd, sj
130 LOCATE my, mx
140 PRINT m$
150 NEXT i
160 END
```

Цикл в строках 100—150 реализует 7 вариантов оператора CIRCLE в строке 120, у которого постоянные параметры: радиус R = 50 и цвет cv = 15. Другие параметры x, y, nd, kd, sj задаются с помощью операторов READ в строке 110 и DATA в строках 30—90. Операторы в строках 130—140 осуществляют вывод на экран нумерации вариантов.

Результат работы программы 8.3 приведен на рис. 8.4.

Закрасить какой-либо замкнутый контур, например нарисованную оператором CIRCLE окружность или эллипс, можно, используя графический оператор PAINT.

PAINT — графический оператор, закрашивающий ограниченную площадь указанным цветом или образом:

```
PAINT (x, y), cvZ, cvR
```

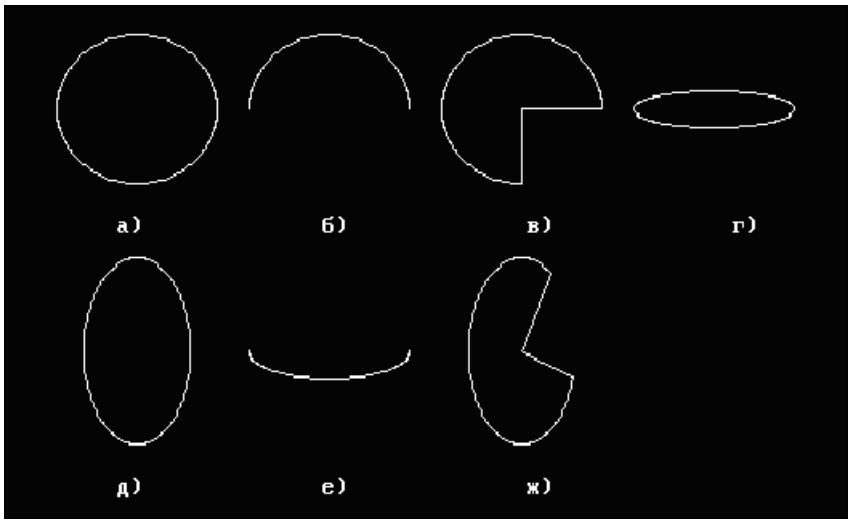


Рис. 8.4. Результаты работы операторов CIRCLE:
(*а*) `CIRCLE (100, 70), 50, 15`;
(*б*) `CIRCLE (220, 70), 50, 15, 0, 3.14`;
(*в*) `CIRCLE (340, 70), 50, 15, -0.01, -4.71`;
(*г*) `CIRCLE (460, 70), 50, 15, , , 0.2`;
(*д*) `CIRCLE (100, 200), 50, 15, , , 1.5`;
(*е*) `CIRCLE (220, 200), 50, 15, 3.14, , 0.3`;
(*ж*) `CIRCLE (340, 200), 50, 15, -1, -6, 1.5`

Здесь `(x, y)` — координаты точки, с которой начинается заливка. Точка может быть указана внутри замкнутого контура или вне его, но не на границе. Если точка находится внутри контура, то закрашивается ограниченная им фигура. Если же точка начала заливки находится вне контура, то закрашивается фон. При попадании на линию контура закрашивания не произойдет.

`cvZ` — цвет заливки, числовое или символьное выражение. Если это числовое выражение, то оно представляет собой атрибут цвета. Если аргумент не указан, то используется атрибут фона.

`cvR` — цвет рамки, числовое выражение, определяющее атрибут цвета замкнутого контура. Если цвет границы указан, то площадь ограничивается линиями этого цвета. Если же аргумент опущен, то используется аргумент цвета заливки `cvZ`.

Обращаем внимание, что контур должен быть замкнут и выполнен линиями одного цвета — цвета рамки `cvR`. Если же в контуре будет хоть одна точка другого цвета, то для BASIC этот контур не будет замкнутым.

Следующая пара операторов нарисует окружность белого цвета (атрибут 15) и закрасит ее ярко-красным цветом (атрибут 12 в операторе `PAINT`):

```
CIRCLE (100, 70), 50, 15
PAINT (100, 70), 12, 15
```

В операторе `PAINT` можно использовать параметр `STEP`, указывающий на то, что координаты точки начала закрашивания задаются как относительные к последней точке рисования. Например:

```
CIRCLE (100, 70), 50, 15
PAINT STEP(0, 10), 12, 15
```

Последней точкой рисования после выполнения оператора CIRCLE будет центр окружности, т. е. точка с координатами x = 100 и y = 70. Тогда начальной точкой закрашивания и последней точкой рисования после выполнения оператора PAINT будет точка с координатами x = 100 и y = 80 (т. к. 70 + 10 = 80).

Рассмотрим следующий оператор из серии "графических примитивов" — оператор LINE.

LINE — графический оператор, рисующий линию или прямоугольник:

```
LINE (xN, yN)-(xK, yK), cv
```

где (xN, yN) — координаты начала линии;

(xK, yK) — координаты конца линии;

cv — номер цвета линии. Если цвет не указан, то линия рисуется основным цветом. Если указан 0, то линия рисуется цветом фона. Если заданы опции B или BF, то прямоугольник рисуется указанным цветом cv.

Оператор:

```
LINE (20, 20)-(120, 35), 1
```

нарисует темно-синюю линию от точки с координатами x = 20 и y = 20 до точки с координатами x = 120 и y = 35.

B — опция, задающая рисование прямоугольника с вертикальными и горизонтальными сторонами:

```
LINE (xN, yN)-(xK, yK), cv, B
```

где (xN, yN) — координаты левого верхнего угла, а (xK, yK) — нижнего правого. Например:

```
LINE (20, 20)-(120, 35), 1, B
```

BF — опция, задающая закрашивание нарисованного прямоугольника указанным цветом cv:

```
LINE (xN, yN)-(xK, yK), cv, BF
```

Например:

```
LINE (20, 20)-(120, 35), 1, BF
```

В операторе LINE можно использовать параметр STEP и привязку к последней точке рисования. Отметим, что последнюю точку можно указать, используя очистку экрана операторами CLS и SCREEN (центр экрана), а также операторами PSET, PRESET, CIRCLE, PAINT, DRAW.

В табл. 8.3 рассмотрены варианты оператора LINE. Пусть последней точкой рисования является точка с координатами x = 20 и y = 30.

Таблица 8.3. *Варианты операторов LINE*

Оператор	Комментарии
`LINE -(70, 70)`	Рисует от (20, 30) до (70, 70)
`LINE -STEP(70, 30)`	Рисует от (20, 30) до (90, 60), т. к. 20 + 70 = 90 и 30 + 30 = 60
`LINE STEP(20, 20)-(120, 90)`	Рисует от (40, 50) до (120, 90), т. к. 20 + 20 = 40 и 30 + 20 = 50
`LINE (20, 20)-STEP(120, 30)`	Рисует от (20, 20) до (140, 50), т. к. 20 + 120 = 140 и 20 + 30 = 50

Таблица 8.3 *(окончание)*

Оператор	Комментарии
`LINE STEP (20, 20)-STEP(80, 20)`	Рисует от (40, 50) до (120, 70), т. к. 20 + 20 = 40 и 20 + 30 = 50 и далее 40 + 80 = 120 и 50 + 20 = 70

Кроме того, в операторе `LINE` можно указать стиль линии, например штриховой:

```
LINE STEP(-100, 35)-STEP(100, 0), 15, , &HF0F0
```

Пример 8.4. Использование оператора *LINE*

Программа демонстрирует возможности оператора `LINE`.

Программа 8.4

```
'8LINE.bas
10 CLS
20 SCREEN 9
'==1===== Блок рисования ========================================
30 LINE (20, 20)-(120, 35), 15
40 LINE STEP(-100, 30)-(120, 80), 15, B
50 LINE (20, 110)-STEP(100, 15), 15, BF
60 LINE STEP(-100, 35)-STEP(100, 0), 15, , &HF0F0
'==2===== Блок вывода нумерации ==================================
70 DATA 6,4,"а)",6,7,"б)",6,10,"в)",6,13,"г)"
80 FOR i = 1 TO 4
90 READ mx, my, m$
100 LOCATE my, mx
110 PRINT m$
120 NEXT i
'================================================================
130 END
```

Для вывода нумерации четырех вариантов оператора `LINE` организован цикл `FOR...NEXT` в строках 80—120. С помощью пары операторов `DATA` в строке 70 и `READ` в строке 90 вводятся нужные значения для `LOCATE` в строке 100 и `PRINT` в строке 110.

Результат работы программы 8.4 дан на рис. 8.5.

Теперь рассмотрим пару операторов `PSET` и `PRESET`, рисующих точку на экране:

```
PSET (x, y), cv
PRESET (x, y), cv
```

где `(x, y)` — координаты точки; `cv` — номер цвета точки.

Если координаты точки находятся вне экрана, то никаких действий не производится и сообщений об ошибке не выдается.

Если в операторе `PSET` не указан цвет, то используется текущий основной цвет. Если в операторе `PRESET` не указан цвет, то тогда точка рисуется цветом фона. В остальном операторы работают одинаково.

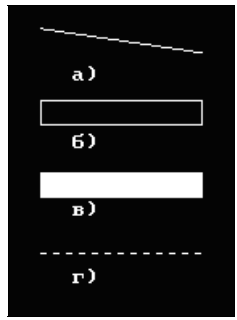


Рис. 8.5. Результат работы операторов LINE:
(*а*) LINE (20, 20)-(120, 35), 15;
(*б*) LINE STEP(-100, 30)-(120, 80), 15, B;
(*в*) LINE (20, 110)-STEP(100, 15), 15, BF;
(*г*) LINE STEP(-100, 35)-STEP(100, 0), 15, , &HF0F0

Пример 8.5. Движение отрезка

В следующей программе оператор PSET рисует отрезок линии из 15 точек, а оператор PRESET стирает ее. В результате действия программы получается движение отрезка.

Программа 8.5

```
'8PSET.bas
10 CLS
20 SCREEN 9
30 FOR x = 0 TO 640 STEP 4
'==1===== Блок рисования отрезка основным цветом ==================
40 FOR i = x TO x + 15
50 PSET (i, 70)
60 NEXT i
'====================================================================
70 FOR iv = 1 TO 10000: NEXT iv                'временная задержка
'==2===== Блок рисования отрезка цветом фона =======================
80 FOR j = x TO x + 15
90 PRESET (j, 70)
100 NEXT j
'====================================================================
110 NEXT x
120 END
```

Цикл FOR...NEXT в строках 40—60 рисует отрезок линии из 15 точек основным цветом и после выдержки времени цикл FOR...NEXT в строках 80—100 рисует такой же отрезок цветом фона, тем самым стирая предыдущий. Внешний цикл FOR...NEXT в строках 30—110 отвечает за движение.

Мощным средством построения графических изображений является оператор DRAW.

DRAW — графический оператор, интерпретирующий символьное выражение и рисующий графический объект:

```
DRAW sv$
```

где sv$ — символьное выражение, одна или более команд рисования.

Оператор объединяет многие возможности других графических операторов в своем макроязыке. Команды макроязыка описывают действия, необходимые для создания образов — движения объектов, углы поворота, цвет, масштаб. Примеры возможных команд приведены в табл. 8.4. Эти команды определяют движение в относительных единицах (кроме команды `MX,Y`, в табл. 8.4 это команды `M20,30` и `BM10,8`). По умолчанию выполняется движение на одну точку. Единица движения модифицируется командой `S`, устанавливающей масштаб. Каждая команда движения задает перемещение относительно текущей графической позиции (последней точки рисования). До выполнения какой-либо команды последняя точка рисования — это центр экрана. Отметим команды способа перемещения:

- `B` — двигаться, но не рисовать (`BM10,8` в табл. 8.4);
- `N` — двигаться, рисовать, вернуться в исходную точку (`NR8` в табл. 8.4).

Отметим некоторые особенности оператора `DRAW`. Выполнение следующего оператора целиком:

```
DRAW "c15 nr2 h2 u5 m-10,-3 u2 r10 u4 m-4,-2 u1 bh2 c0 nr3 bf2"
```

или разбитого на два оператора:

```
DRAW "c15 nr2 h2 u5 m-10,-3 u2 r10"
DRAW "u4 m-4,-2 u1 bh2 c0 nr3 bf2"
```

приведет к одинаковому результату.

Команды в кавычках лучше писать раздельно, но можно и слитно, большими буквами или маленькими. Это позволяет группировать команды для удобства редактирования. Например, в следующем операторе:

```
DRAW "c8bm447,0g3l2g2 F1 R3 D1 G3 l1 G4L1G3L1 g4 l3 g2 d3 g2 d3"
```

четыре группы команд: маленькими буквами слитно, большими буквами раздельно, большими буквами слитно, маленькими буквами раздельно. Варьируя подобные группы, можно отмечать поворотные места в операторе. Основные команды перемещения нетрудно запомнить с помощью рис. 8.6.

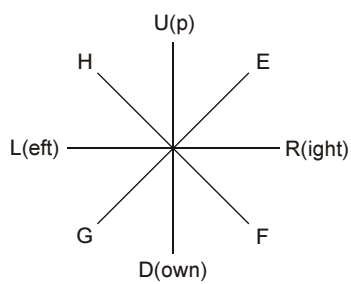


Рис. 8.6. Направления перемещения и их обозначения

Таблица 8.4. *Примеры команд оператора* `DRAW`

Команда	Описание
`U10`	Вверх на 10
`D7`	Вниз на 7
`L15`	Влево на 15
`R22`	Вправо на 22

Таблица 8.4 *(окончание)*

Команда	Описание
E11	Вправо вверх на 11
F5	Вправо вниз на 5
G7	Влево вниз на 7
H6	Влево вверх на 6
M20,30	Линия от последней точки до точки с координатами x = 20, y = 30
M+5,-10	Линия от последней точки (пусть x = 20, y = 20) до точки с координатами x = 25, y = 10 (шаг)
BM10,8	Переход, не рисуя, в точку с координатами x = 10, y = 8
C2	Установка цвета 2 (зеленый)
P1,2	Закрашивание замкнутой фигуры, где 1 (синий) — цвет фигуры, 2 (зеленый) — цвет границы
S7	Установка масштаба 7 (параметр масштаба может быть от 1 до 255). Увеличивает единицы перемещения команд U, D, L, R, M
A2	Установка угла поворота 2 (на 180°). 0 — на 0°, 1 — на 90°, 3 — 270°
TA30	Поворачивает изображение на угол в 30° (угол поворота в пределах от –360° до 360°). Если угол > 0, то поворот против часовой стрелки, иначе — по часовой стрелке
NR8	Рисует вправо на 8 и возвращается назад в исходную точку (буква N)

Примечание

1. Перед командой Pn,n необходимо перейти без рисования внутрь закрашиваемой фигуры.
2. Цвет по ходу рисования можно неоднократно переустанавливать командой Cn.
3. Замкнутый контур не будет замкнутым для Pn,n, если линия границы контура содержит хотя бы одну точку другого цвета.

Возможно использование операторов и следующего вида:

```
DRAW "c8 bm447,0 g3 l2 g2" + m1$ + m2$ + "g4 l3 g2 d3 g2 d3"
```

где m1$ и m2$ — некоторые символьные выражения.

Для многократного вызова на экран дисплея изображения существует подкоманда "x" + VARPTR$(n$). Например, следующий фрагмент программы:

```
n$ = "c14 u40 l40 d40 r40"
DRAW "x" + VARPTR$(n$ )
DRAW "bm5,5 x" + VARPTR$(n$)
```

выведет на экран два прямоугольника: один в середине экрана, а другой в левом верхнем углу.

При работе с графикой полезной будет и графическая функция POINT, читающая номер цвета точки экрана или возвращающая координаты точки:

```
POINT (x, y)
```

где (x, y) — координаты графической точки, цвет которой надо узнать.

Если указанная точка не находится на экране, то POINT возвращает –1.

Пример 8.6. Построение зеркального изображения

Требуется разработать программу рисования зеркального изображения.

Программа 8.6

```
'8zerRis.bas    Зеркальное изображение
10 CLS
20 SCREEN 9
'==1===== Блок рисования фигуры ===================================
30 x = 200
40 PSET (x, 300), 0
50 DRAW "c11 u100 r30 u7 r10 u3 l42 m-5,-8 m+4,-1 e3 u4 m-1,-3 h3 m-3,-1 l5"
60 DRAW "m-5,+2 g6 d40 r3 d5 m-5,+10 m-10,+70 nr35 be1 p11,11"
70 GOSUB 500
80 GOTO 600
'==2===== Подпрограмма рисования зеркального изображения ===========
500 xfn = x - 40
510 yfn = 166
520 FOR yf = yfn TO yfn + 135
530 FOR xf = xfn TO xfn + 80
540 PSET (600 - xf, yf), POINT(xf, yf)
550 NEXT xf, yf
560 RETURN
'====================================================================
600 END
```

Центральным в программе является оператор `PSET` в строке 540, являющейся телом двойного цикла в строках 520—550, который захватывает на экране прямоугольник по `x` от 160 до 240 и по `y` от 166 до 201. В этой области экрана и находится изображение, которое нужно зеркально отобразить. С помощью оператора `POINT` в строке 540 последовательно определяется цвет каждой точки указанной области экрана, который и является аргументом цвета оператора `PSET` в строке 540, рисующего точки в зеркальной области экрана. Осью симметрии является вертикальная ось `x` = 300.

Результат работы программы 8.6 дан на рис. 8.7.

Рис. 8.7. Построение зеркального изображения

Предлагаем читателю самостоятельно разработать программу, строящую зеркальное изображение относительно горизонтальной оси симметрии.

Оператор POINT, в котором указан один аргумент, позволяет определить координаты графического курсора:

```
POINT (n)
```

где n — числовое выражение от 0 до 3, указывающее один из четырех вариантов определения текущей позиции графического курсора. Эти варианты представлены в табл. 8.5.

Таблица 8.5. *Значения аргумента оператора* POINT

Аргумент	Возвращаемое значение
0	Физическая *x*-координата
1	Физическая *y*-координата
2	Относительная *x*-координата
3	Относительная *y*-координата

Примечание

Если не использовался оператор WINDOW, то оператор POINT с аргументом 2 возвращает то же значение, что и с аргументом 0, а с аргументом 3 — то же, что и с аргументом 1.

При построении графических изображений для определения последней точки рисования можно использовать следующую строку кода:

```
PRINT "x = "; POINT(0); "y = "; POINT(1)
```

которая выводит на экран координаты последней точки.

Операторы GET и PUT будут рассмотрены в *разд. 8.3*.

8.2. Статическая графика

Начинать программу следует с оператора CLS. Установка экранного режима осуществляется операторами SCREEN и COLOR. Авторы обычно используют SCREEN 9.

Изображение разбивается на фрагменты, рисуемые с помощью графических примитивов: точек (операторы PSET, PRESET), линий, прямоугольников и закрашенных прямоугольников (оператор LINE), окружностей, эллипсов и дуг окружностей и эллипсов (оператор CIRCLE). Более сложные контуры рисуются с использованием оператора DRAW. Рекомендуется нарисованные контуры сразу же закрашивать (оператор PAINT), соблюдая следующие правила:

- контур должен быть **замкнут**, в нем не должно быть прокола хотя бы на одну экранную точку;
- контур должен быть **одноцветен**, разноцветность составляющих замкнутого контура для BASIC равноценна разрыву;
- координаты точки начала закраски должны лежать **внутри контура**.

Если точка закраски попала вне контура, то закрасится весь экран за исключением самого контура. Если точка закраски попала на контур или на точку с цветом контура, то ничего не закрасится.

Для проверки попадания точки начала закраски внутрь контура рекомендуется использовать оператор PSET, рисующий точку ярким контрастным цветом, после чего заменить его на PAINT с теми же координатами.

Для закрашивания окружностей и эллипсов целесообразно использовать:

```
PAINT STEP(0, 0), cvZ, cvR
```

Перед созданием программы начертите рисуемое изображение в масштабе на бумаге в клетку. Тогда, используя выводимую на экран координатную сетку, нетрудно будет запрограммировать это изображение. Дополнительно к сетке неплохо использовать строку кода:

```
PRINT "x = "; POINT(0); "y = "; POINT(1)
```

выводящую на экран координаты последней точки.

Необходимо определить координаты начальной (опорной) точки, привязав к ней фрагменты изображения, используя относительные координаты (графические операторы с параметром STEP либо выражения для координат типа: x = x + dx и y = y + dy). Тогда можно передвигать рисунок по экрану без изменения программы, меняя лишь координаты опорной точки.

Определите последовательность построения и закраски замкнутых контуров, нередко она имеет большое значение. Рисуйте контрастным, хорошо различимым на фоне цветом, его всегда потом можно изменить. Контролируйте свои шаги при создании программы, постоянно запуская ее после введения новой команды оператора DRAW или написания очередных операторов LINE, CIRCLE и пр. Если шаг был удачным, сохраните его, чтобы иметь в запасе, на всякий случай, выход без сохранения. Старайтесь, чтобы текст программы не выходил за боковые границы экрана. Например, оператор DRAW всегда можно непосредственно продолжить новым оператором DRAW на новой строке. В программе 8.7 приведен вариант координатной сетки, который показан на рис. 8.8.

Программа 8.7

```
'8cetka.bas    Координатная сетка
10 CLS
20 SCREEN 9
'30 COLOR , 2
'===== Сетка ========================================================
'40 GOTO 100
50 FOR xc = 0 TO 650 STEP 10
60 IF xc MOD 100 = 0 THEN k1 = 9 ELSE k1 = 15
70 LINE (xc, 0)-(xc, 370), k1
80 LINE (0, xc)-(650, xc), k1
90 NEXT xc
100 SLEEP
'==1===== Блок рисования ============================================
LOCATE 23, 60: PRINT "x = "; POINT(0); "y = "; POINT(1)
600 END
```

Оператор LINE в строке 70 рисует вертикальные линии сетки, а LINE в строке 80 — горизонтальные. Цвет линий сетки задается в строке 60. Причем все сотые линии выделяются другим цветом (по условию xc MOD 100 = 0). В представленном варианте линии сетки белые, а сотые — синие. Цвет линий сетки легко можно изменить. Такая необходимость изменения цвета вызвана тем, что при совпадении цвета линий сетки и контура рисуемой фигуры, ли-

нии сетки будут мешать закрасить фигуру. Оператор в строке 40 позволяет вывести из рассмотрения сетку, не удаляя ее из программы. Оператор PRINT в предпоследней строке программы выводит на экран координаты последней точки рисования. Место вывода этой информации также можно изменить, меняя параметры в операторе LOCATE.

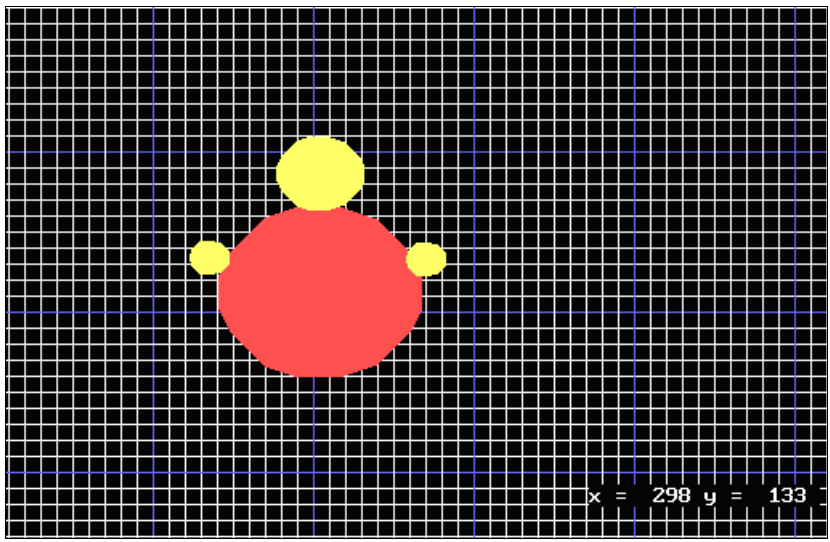


Рис. 8.8. Вид экрана с координатной сеткой

После завершения разработки программы сетка и строка с указанным оператором PRINT благополучно удаляются.

Теперь поясним на примере 8.7 понятие начальной или опорной точки изображения.

Пример 8.7. Рисование лампочки

Требуется разработать программу рисования электрической лампочки.

Программа 8.8

```
'81ampa.bas    Электрическая лампочка накаливания
10 CLS
20 SCREEN 9
30 LINE (0, 30)-STEP(640, 30), 14, BF
'==1===== Блок рисования лампочки ================================
40 CIRCLE (300, 100), 30, 11: PAINT STEP(0, 0), 11, 11       'колба
50 DRAW "bm-30,-6 m+16,-34 r28 nm+16,+34 bd10 p11,11"        'колба
60 DRAW "c1 bu10 u10 h4 l20 g4 d10 r28 bm-12,-2 p1,1"        'цоколь
70 LINE STEP(0, 3)-STEP(10, 45), 1                  'держатель нити
80 LINE STEP(-26, 0)-STEP(10, -45), 1               'держатель нити
90 LINE STEP(-8, 40)-STEP(23, 0), 1                          'нить
'==================================================================
100 END
```

Опорной или начальной точкой в программе 8.8 является центр окружности колбы, рисуемый оператором в строке 40.

Результат работы программы 8.8 дан на рис. 8.9.

Перед началом разработки программы целесообразно наметить порядок рисования. Нагляднее всего это будет сделать в виде блок-схемы (рис. 8.10).

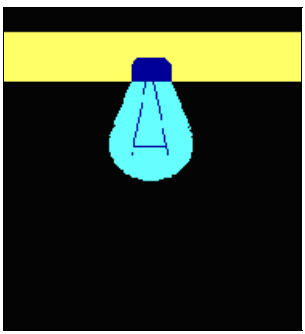


Рис. 8.9. Лампочка накаливания

Рис. 8.10. Блок-схема алгоритма рисования лампочки

Покажем теперь, как благодаря опорной точке можно ввинтить несколько лампочек без особых затрат.

Пример 8.8. Рисование нескольких лампочек

Программа рисования нескольких электрических лампочек.

Программа 8.9

```
'81ampa2.bas    Лампочки
10 CLS
20 SCREEN 9
30 LINE (0, 30)-STEP(640, 30), 14, BF
'==1===== Блок расстановки лампочек ================================
40 FOR x = 100 TO 500 STEP 100
50 GOSUB 500                          'рисование очередной лампочки
60 NEXT x
70 GOTO 600
'==2===== Подпрограмма рисования лампочки ==========================
500 CIRCLE (x, 100), 30, 11: PAINT STEP(0, 0), 11, 11       'колба
510 DRAW "bm-30,-6 m+16,-34 r28 nm+16,+34 bd10 p11,11"      'колба
```

```
520 DRAW "c1 bu10 u10 h4 l20 g4 d10 r28 bm-12,-2 p1,1"      'цоколь
530 LINE STEP(0, 3)-STEP(10, 45), 1                  'держатель нити
540 LINE STEP(-26, 0)-STEP(10, -45), 1               'держатель нити
550 LINE STEP(-8, 40)-STEP(23, 0), 1                         'нить
560 RETURN
'=============================================================
600 END
```

Результат работы программы 8.9 показан на рис. 8.11.

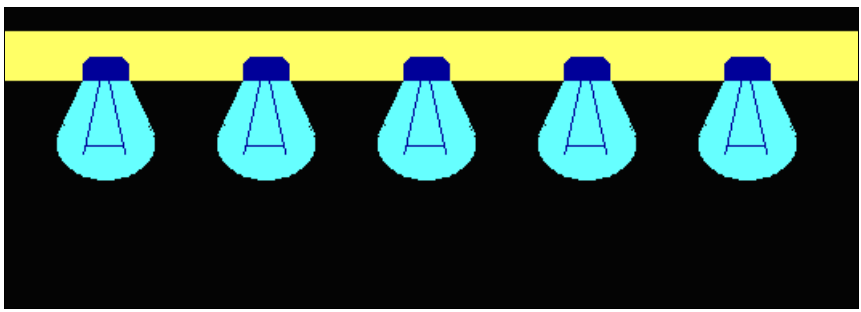


Рис. 8.11. Результат программы рисования нескольких лампочек

Блок рисования лампочки оформляется как подпрограмма, причем изменения касаются лишь оператора `CIRCLE`. Для изменения координаты вывода лампочки организован цикл `FOR...NEXT` в строках 40—60. В этой программе выводятся 5 лампочек. Если же необходимо просто передвинуть одну лампочку, то цикл `FOR...NEXT` не нужен, а просто задаются необходимые координаты в операторе `CIRCLE` в строке 500.

В *главе 1* был пример 1.4 с гаммой. Покажем несколько измененный вариант проигрывания гаммы.

Пример 8.9. Гамма

Следующая программа рисует 7 клавиш пианино, нажатие которых сопровождается соответствующими звуками. Гамма проигрывается туда и обратно.

Программа 8.10

```
'8pianino.bas   Пианино
10 CLS                              'очистка экрана
20 SCREEN 9                         'установка экранного режима
30 COLOR 14, 4            'установка экранных цветов (15 — цвет фона)
40 DIM nota$(8)                     'готовит в памяти место для нот
'==1===== Блок рисования клавиш пианино ==========================
50 LINE (100, 80)-STEP(400, 150), 15, BF
60 LINE -STEP(-400, -15), 7, BF
    '-------------------------------------------------------------
70  FOR j = 1 TO 4
80  IF j < 4 THEN in = 1: ik = 8: p = 1 ELSE in = 7: ik = 1: p = -1
90  IF j = 3 THEN  ELSE 110
100  PRINT " Для воспроизведения гаммы нажмите клавишу Enter": SLEEP
```

```
110  FOR i = in TO ik STEP p
120 SELECT CASE j
CASE 1: READ nota$(i)
     DATA "n1","n2","n3","n4","n5","n6","n7","n8"
CASE 2: LINE (50 + i * 50, 215)-STEP(50, -135), 7, B
CASE 3, 4: LINE (51 + i * 50, 225)-STEP(48, -35), 15, BF
                  n$ = nota$(i)
                  PLAY "x" + VARPTR$(n$)
         FOR iv = 1 TO 100000: NEXT iv
         LINE (100, 230)-STEP(400, -15), 7, BF
130 END SELECT
140 NEXT i, j
'=================================================================
150 END
```

На рис. 8.12 показан вывод программы 8.10.

Рис. 8.12. Пианино с нажатой клавишей фа

Программа 8.10 уже с элементами движения, поскольку клавиши то нажимаются, то отпускаются. В программе 4 раза используется вложенный цикл `FOR...NEXT` в строках 110—130 с разными параметрами. Первый раз — для ввода нот, второй раз — для рисования линий, разделяющих клавиши, третий раз — для проигрывания гаммы и нажатия клавиш и четвертый — для проигрывания гаммы в обратном порядке. Перебор названных вариантов обеспечивает внешний цикл `FOR...NEXT` в строках 70—130. Обрабатывает варианты условный оператор `SELECT CASE` в строках 120—130. Соответствующее изменение параметров вложенного цикла `FOR...NEXT` задает в строке 80 условный оператор `IF...THEN`.

8.3. Динамическая графика (анимация)

Принцип создания движения — вывод на экран изображения, которое после определенной временной задержки стирается и немедленно рисуется в другом положении, смещенном относительно предыдущего в соответствии с заданной траекторией движения. Траектория движения определяется программой. В зависимости от того, как выводится и стирается изображение, различаются следующие способы:

1. Выводить изображение различными цветами переднего плана, а стирать, выводя то же самое изображение цветом фона.
2. Выводить изображение так же, как в первом способе, а стирать — оператором CLS.
3. Выводить изображение так же, как в первом способе, а стирать — прямоугольником, цвет контура и заливки которого совпадает с цветом фона.
4. Выводить изображение как в пункте один, а стирать задние планы изображения цветом фона.
5. Записать изображение в буфер, размещаемый в динамической памяти, а затем выводить его в нужных местах экрана. Стирать повторным наложением изображения из буфера либо способом 2 или 3.

Существуют и другие способы, здесь не приводимые. Наиболее качественное изображение получается 4-м способом, но он достаточно легко реализуется лишь в случае прямолинейного движения.

Для реализации 5-го способа используются операторы GET и PUT.

GET — графический оператор, считывающий графическое изображение с экрана в массив:

```
GET (x1, y1)-(x2, y2), im
```

где x1, y1, x2, y2 — координаты прямоугольной области экрана, где находится считываемое изображение: (x1, y1) — координаты левого верхнего угла; (x2, y2) — координаты правого нижнего угла.

im — имя массива для записи изображения. Массив может быть любого числового типа, а его размер должен вместить изображение.

В операторе GET допускается использовать параметр STEP, указывающий на то, что координаты берутся относительно последней нарисованной точки. То есть возможны варианты:

```
GET STEP(x1, y1)-(x2, y2), im
GET (x1, y1)-STEP(x2, y2), im
GET STEP(x1, y1)-STEP(x2, y2), im
```

Оператор GET считывает нужное изображение с экрана в массив, а оператор PUT трансформирует изображение из массива на экран.

Размер массива в байтах вычисляется по формуле:

```
4 + INT(((x2 – x1 +1) * k1 + 7) / 8) * k2 * ((y2 – y1) + 1)
```

Число бит на точку в слое и число слоев зависят от спецификаций оператора SCREEN. В табл. 8.6 показаны эти значения для каждого режима экрана.

Таблица 8.6. *Характеристики спецификаций SCREEN*

Режим экрана	Битов на точку в одном слое (k1)	Число слоев (k2)	Режим экрана	Битов на точку в одном слое (k1)	Число слоев (k2)
SCREEN 1	2	1	SCREEN 10	1	2
SCREEN 2	1	1	SCREEN 11	1	1
SCREEN 7	1	4	SCREEN 12	1	4
SCREEN 8	1	4	SCREEN 13	8	1
SCREEN 9	1	4			

Примечание

Количество байт на элемент в числовых массивах в зависимости от типа следующее: 2 — для целого типа, 4 — для длинных целых и обычной точности, 8 — для чисел двойной точности.

Значит, для SCREEN 9 формула будет следующего вида:

```
4 + INT(((x2 - x1 + 1) * 1 + 7) / 8) * 4 * ((y2 - y1) + 1)
```

PUT — графический оператор, рисующий на экране изображение, взятое в массив оператором GET:

```
PUT (x, y) , im, d
```

где (x, y) — координаты левого верхнего угла прямоугольника, в котором будет выводиться изображение;

im — имя массива с изображением;

d — параметр, позволяющий накладывать образ на экран со специальными эффектами. По умолчанию — XOR. Возможные варианты параметра описаны в табл. 8.7.

Таблица 8.7. *Варианты параметра "действие" оператора* PUT

Параметр	Комментарии
PSET	Переводит изображение точка за точкой на экран. Каждая точка имеет тот же цветовой атрибут, который был при считывании массива оператором GET
PRESET	Так же, как и PSET, но выдает инверсное изображение ("негатив")
AND	Изображение накладывается на существующую картинку. Получается результат логического умножения, при котором имеющие одинаковый цвет точки сохраняются, а остальные меняют цвет
OR	Изображение накладывается на существующую картинку. Получается результат логического сложения между образом и картинкой. Новое изображение не стирает предыдущего
XOR	Действует так, что точки на экране, имеющие тот же цвет, что и образ, инвертируются. При помещении образа на одно и то же место дважды предыдущая картинка восстанавливается, что позволяет двигать образ по экрану, не стирая фона

Если вы предполагаете рисовать довольно часто, то авторы рекомендуют в качестве стартового варианта использовать сетку, дополненную заготовками наиболее используемых операторов. Тогда новую разрабатываемую программу можно формировать путем копирования и редактирования. В программе 8.11 приведен пример сетки с заготовками.

Программа 8.11

```
'cetka.bas    Координатная сетка
CLS
SCREEN 9
'COLOR , 2
'===== Сетка ========================================================
'GOTO 60
FOR xc = 0 TO 650 STEP 10
IF xc MOD 100 = 0 THEN k1 = 9 ELSE k1 = 15
```

```
LINE (xc, 0)-(xc, 370), k1
LINE (0, xc)-(650, xc), k1
NEXT xc
60 SLEEP
'==1===== Блок рисования ==========================================
LOCATE 23, 60: PRINT "x = "; POINT(0); "y = "; POINT(1)
600 END

'DRAW "bm320,175"
'DRAW "c14 r4 m+3,-1 e3 m+1,-2 U3 M-1,-2 H3 M-3,-1 l4 m-3,+1 g3 m-1,+2"
'DRAW "D3 M+1,+2 F3 M+3,+1 bu1 p14,14"
'CIRCLE (x, y), R, cvet
'CIRCLE (x, y), 10, 14: PAINT STEP(0, 0), 14, 14
'CIRCLE (x, y), 20, 7, 1.57, 4.71, 1.5
'DRAW "bm320,170 x" + VARPTR$(n$)
'FOR iv = 1 TO 1000: NEXT iv     'выдержка времени
'RANDOMIZE 32767 — TIMER
'PSET (xl, yl + 20 * (i — 1)), 0: DRAW "ta45 x" + VARPTR$(cv0$)
'IF INKEY$ = CHR$(27) THEN EXIT DO   'выход из цикла (клавиша Esc)
'==2===== Снятие изображения воздушного шара ======================
'B = INT((40 * 2 + 7) / 8 * 55)
'DIM bug(B)
'GET (30, 255)-(70, 309), bug
'LINE (30, 255)-(70, 310), 0, BF
'==4===== Блок движения объекта ===================================
'x = 95: y = 65: av = -1
'DO UNTIL INKEY$ = CHR$(27)      'клавиша Esc
'DO
'GOSUB 500
'FOR iv = 1 TO 10000: NEXT iv    'выдержка времени
'IF x = 22 THEN EXIT DO          'выход из цикла
'CLS                             'стирает
'x = x — 20
'LOOP
'LOOP
'================================================================
'SELECT CASE av
'CASE -1:
'CASE 1:
'END SELECT
'================================================================
```

Не забудьте после размещения необходимого оператора в программе убрать апостроф в начале строки. После окончания разработки программы все лишнее следует убрать.

Подход к организации движения зависит от сюжета. Если необходимо привести в движение простые предметы, такие как мячик, стрела и даже более сложные — яхта или лебедь и тому подобное, то их перемещение организуется путем последовательного изменения координат, не меняя самого изображения. В других случаях требуется построить несколько изображений, и, поочередно меняя их, тем самым и добиваются эффекта движения. Причем для получения нескольких изображений можно менять лишь фрагменты основной фигуры. Так, в своей программе прыгающей девочки студентка СПбГУСЭ Светлана Зуйкова достигает нужного эффекта, меняя лишь наклон головы.

Пример 8.10. Прыгающая девочка

Следующая программа рисует прыгающую девочку.

Программа 8.12

```
'8girl.bas     Прыгающая девочка
10 SCREEN 12
20 x = 520
30 Yd = 200
40 a = 1
50 FOR J = 1 TO 91
60 CLS
70 IF a = -1 THEN y = Yd ELSE y = Yd + 10
80 CIRCLE (x, y), 40, 2, , , 3: PAINT STEP(0, 0), 2, 2      'блузка
90 CIRCLE STEP(20, -30), 25, 6, , , .25: DRAW "p6,6"        'левая рука
100 CIRCLE STEP(-40, 0), 25, 6, , , .25: DRAW "p6,6"        'правая рука
110 CIRCLE STEP(20, -25), 15, 6: PAINT STEP(0, 0), 6, 6     'голова
120 CIRCLE STEP(-10, 115), 25, 6, , , 3: DRAW "p6,6"        'правая нога
130 CIRCLE STEP(20, 0), 25, 6, , , 3: DRAW "p6,6"           'левая нога
140 DRAW "c12 bm-30,-40 r40 m+20,+30 l80 m+20,-30 bf10 p12,12"   'юбка
150 SELECT CASE a
       CASE -1: PSET (x - 5, y - 48), 0                     'глаза
   PSET STEP(10, 0), 0
                     LINE STEP(-5, 5)-STEP(0, 1), 0        'нос
        CASE 1:  LINE (x - 2, y - 45)-STEP(4, 0), 0         'рот
                       PSET STEP(-7, -13), 0                'глаза
    PSET STEP(10, 0), 0
                      LINE STEP(-5, 5)-STEP(0, 1), 0        'нос
160 END SELECT
170 PSET (x - 8, Yd - 64), 1
180 DRAW "m+16,+8 u8 m-16,+8 u8 bf4 p1,1 br8 p1,1"          'бант
190 FOR IV = 1 TO 50000: NEXT IV                  'выдержка времени
200 a = -a    'a - переключатель положений: a=1 - стоит, a=-1 - прыгает
210 x = x - 5
220 NEXT J
230 END
```

Разные положения этой девочки-резвушки можно увидеть на рис. 8.13.

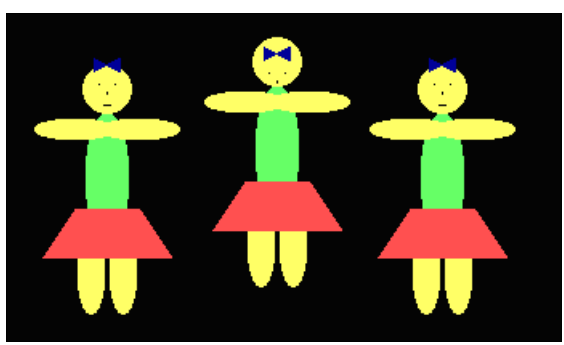


Рис. 8.13. Прыгающая девочка

Оператор IF...THEN в строке 70 определяет значение координаты y в зависимости от значения переключателя a. Оператор SELECT CASE в строке 150 обрабатывает значение a, выводя соответствующий наклон головы девочки. Цикл FOR...NEXT в строках 50—220 обеспечивает горизонтальное перемещение девочки.

Другой способ организации движения — вращение вокруг опорной точки. Для этого фигура рисуется с помощью оператора DRAW, а вращение обеспечивается изменением угла поворота φ фигуры посредством команды TAφ (см. табл. 8.4). Отметим, что центр вращения может как принадлежать вращаемой фигуре, так и находиться вне перемещающегося изображения. Проиллюстрируем предлагаемый способ на примере.

Пример 8.11. Вращение вокруг опорной точки

Требуется разработать программу рисования динамической картинки известного армейского сюжета: упал — отжался.

Программа 8.13

```
 '8upalOt.bas    Упал — отжался
10 CLS
20 SCREEN 9
30 COLOR , 14
'==1===== Блок рисования солдата ==================================
40 s1$ = "c8 nl5 d6 l15 u30 r15 nm-5,+25 bg10 p8,8 C2 BE10 NL15"   'сапог
50 s2$ = "M+5,-30 R3 U40 M-8,-10 L10 M-8,+10 D40 R3 ND30 BU5 P2,2" 'тело
60 s3$ = "c6 nl3 r23 u5 l26 nd5 bf3 p6,6"                          'ремень
70 s4$ = "bm+5,-43 c14 h5 u8 e5 r10 f5 d8 g5 l10 be10 p14,14"    'голова
80 s$ = s1$ + s2$ + s3$ + s4$
'==2===== Блок "упал" ============================================
90 DO
100 ss = ss + 1
110 IF ss > 7 THEN EXIT DO
120 CLS
130 SELECT CASE ss
       CASE 1: DRAW "bm245,254 x" + VARPTR$(s$)
                    SLEEP 2
        CASE 2: DRAW "bm245,254 ta-15 x" + VARPTR$(s$)
        CASE 3: DRAW "bm245,254 ta-30 x" + VARPTR$(s$)
        CASE 4: DRAW "bm245,254 ta-45 x" + VARPTR$(s$)
        CASE 5: DRAW "bm245,254 ta-60 x" + VARPTR$(s$)
        CASE 6: DRAW "bm245,254 ta-75 x" + VARPTR$(s$)
        CASE 7: DRAW "bm245,254 ta-90 x" + VARPTR$(s$)
140 END SELECT
150 FOR iv = 1 TO 10000: NEXT iv
160 LOOP
'==3===== Блок "отжался" ==========================================
170 DO
    '------------ положение "лежа" -------------------------------
180 CLS
190 DRAW "bm245,254 ta-90 x" + VARPTR$(s$)                  'солдат
200 DRAW "c10 bm-10,+20 nm+6,+4 d8 m+6,+6 m+12,-6 u8 nm-12,+4" 'рука
210 LINE STEP(3, 0)-STEP(-12, 3), 14, BF                    'ладонь
    '-------------------------------------------------------------
```

```
220 IF INKEY$ = CHR$(27) THEN EXIT DO                'выход из цикла
230 FOR iv = 1 TO 50000: NEXT iv                  'временная задержка
    '------------ положение "в упоре" ----------------------------
240 CLS
250 DRAW "bm245,254 ta-75 x" + VARPTR$(s$)                   'солдат
260 DRAW "c10 bm-10,+20 m-3,+8 m+40,-0 m+3,-8 nm-40,-0 bg5 p2,10" 'рука
270 LINE STEP(-1, 2)-STEP(12, 3), 14, BF                     'ладонь
280 FOR iv = 1 TO 50000: NEXT iv                  'временная задержка
290 LOOP
'================================================================
300 END
```

На рис. 8.14 показаны основные положения солдата при этом ритуальном действии.

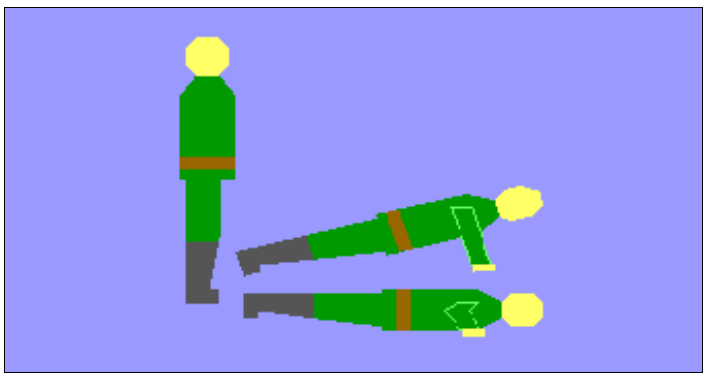


Рис. 8.14. Упал — отжался

Движение состоит из двух фаз. Первая фаза — "упал", заключающаяся в перемещении фигуры солдата из вертикального положения в горизонтальное. В программе обеспечивается циклом `DO...LOOP` в строках 90—150, причем перебор углов поворота командой `TAφ` организован с помощью условного оператора `SELECT CASE`. Вторая фаза — "отжался", заключающаяся в попеременном выводе солдата то в положении "лежа", то в положении "в упоре". Обеспечивается циклом `DO...LOOP` в строках 170—290.

В следующем примере к вращательному движению фигуры добавляется прямолинейное движение пули.

Пример 8.12. Прямолинейное движение и вращение

Требуется разработать программу рисования динамической картинки "выстрелом в сердце разбудишь меня".

Программа 8.14

```
'8victrl.bas    Выстрел
10 CLS
20 SCREEN 9
30 COLOR 1, 15
40 PRINT : PRINT "    ВЫСТРЕЛОМ В СЕРДЦЕ РАЗБУДИШЬ МЕНЯ"
```

```
'==1===== Блок ввода координат ====================================
50 DIM un$(20)
60 DATA "90","80","70","60","50","40","30","20","10","0"
70 FOR i = 1 TO 10: READ un$(i): NEXT i
'==2===== Блок рисования девушки ==================================
80 CIRCLE (450, 115), 20, 14: PAINT STEP(0, 0), 14, 14          'Лицо
90 CIRCLE STEP(0, 30), 20, 13: PAINT STEP(0, 0), 13, 13         'Блузка
100 LINE STEP(-22, 0)-STEP(44, 35), 13, BF
     '----- Волосы ---------------------------------------------
110 CIRCLE STEP(-22, -65), 22, 6, -.01, -3.14: PAINT STEP(0, -2), 6, 6
120 LINE STEP(22, 2)-STEP(-22, 20), 6, BF
     '-------------------------------------------------------------
130 LINE STEP(22, 46)-STEP(-44, 30), 1, BF                      'Юбка
140 DRAW "c14 ta0 s4 br4 nr40 m+10,+70 r20 nm+6,-70 bh5 p14,14" 'Брюки
150 DRAW "c6 bm+3,+6 d10 l5 u4 m-18,+4 u5 m+8,-5 r15 bg5 p6,6"  'Туфли
160 DRAW "bm-27,-130 c13 u10 m-60,+27 d8 nm+60,-18 be3 p13,13"  'Рука
170 DRAW "c8 bm-3,-5 m-20,+7 m-2,-3 m+20,-7 m+2,+3 bl3 p8,8"    'Ствол
180 DRAW "c14 br2 nm-5,+3 d8 l5 nu6 be2 p14,14"                 'Кулак
'==3===== Блок рисования юноши ====================================
190 s1$ = "C6 D10 R30 U5 M-10,-5 L19 BF5 P6,6 c8 bm-19,-106 nr44
                  m+12,+100"
200 s2$ = "r20 m+12,-100 bg10 p8,8 be10 nl44 u50 m-5,-8 h6 m-7,-4
                  l10 m-7,+4"
210 s3$ = "g6 m-4,+8 nd50 br10 p8,8 c14 bm+8,-18 s8 r4 m+3,-1 e3
                  m+1,-2 U3"
220 s4$ = "M-1,-2 H3 M-3,-1 l4 m-3,+1 g3 m-1,+2 D3 M+1,+2 F3 M+3,+1 bu1
                  p14,14"
230 s0$ = s1$ + s2$ + s3$ + s4$
'==4===== Блок движения юноши ====================================
240 FOR j = 1 TO 10
250 s$ = "ta" + un$(j) + s0$
260 DRAW "bm300,282 s4 x" + VARPTR$(s$)
270 IF e = 0 THEN e = 1: GOSUB 500          'подпрограмма выстрела
280 FOR iv = 1 TO 10000: NEXT iv            'временная задержка
290 IF j = 10 THEN EXIT FOR
300 LINE (0, 80)-STEP(333, 350), 0, BF
310 NEXT j
320 GOTO 600
'==5===== Подпрограмма выстрела ===================================
500 x = 342: y = 179: SLEEP 1
510 DO
520 LINE (x, y)-STEP(3, 3), 12, BF          'рисует пулю
530 IF ev = 0 THEN ev = 1: PLAY "mb n1"     'звук выстрела
540 FOR iv = 1 TO 3000: NEXT iv             'временная задержка
550 LINE (x, y)-STEP(3, 3), 0, BF           'стирает пулю
560 IF y >= 256 THEN EXIT DO
570 x = x — 1: y = y + .35
580 LOOP
590 RETURN
'==================================================================
600 END
```

На рис. 8.15 запечатлены 4 фрагмента сюжетной линии программы 8.14. На первом фрагменте пуля уже выпущена, но сердца еще не достигла, и юноша еще спит. На трех последующих фрагментах он уже проснулся и даже поднялся, соответственно, на 30°, 60° и 90°. Если в предыдущей программе 8.13 угол поворота фигуры задан в явном виде, то в данной программе он формируется в строке 250 из элементов массива `un$(j)`.

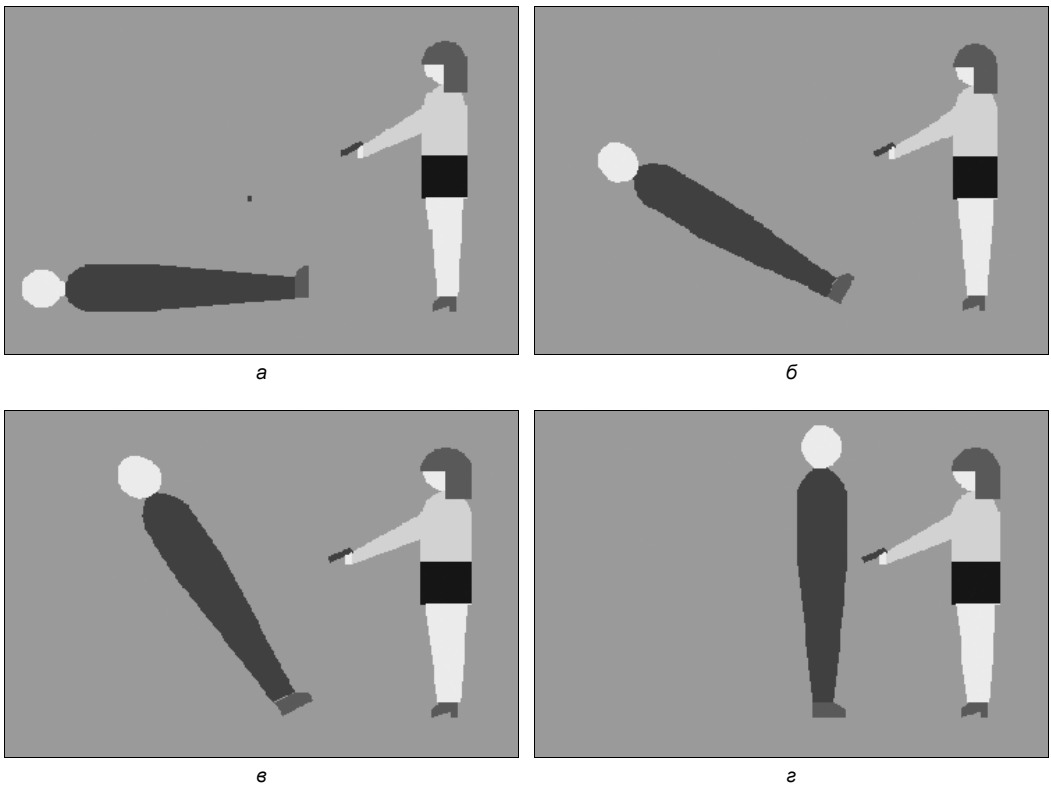


Рис. 8.15. Четыре фазы побудки выстрелом в сердце

Если строку 250 изменить следующим образом:

```
250 s$ = "ta" + STR$(100 — j * 10) + s0$
```

то можно исключить блок ввода координат в строках 50—70.

В рассмотренных примерах единовременно осуществляется движение лишь одного объекта. Если же одновременно перемещаются несколько объектов, то это можно организовать, задавая каждому свои переменные для координат (`x1`, `y1`; `x2`, `y2`; `x3`, `y3` и т. д.). Следовательно, и приращение координат будет свое для каждого из движущихся объектов:

```
x1 = x1 + dx1: y1 = y1 + dy1
x2 = x2 + dx2: y2 = y2 + dy2
x3 = x3 + dx3: y3 = y3 + dy3
```

и т. д.

Поясним это на следующем примере.

Пример 8.13. Движение нескольких объектов одновременно

Требуется разработать программу рисования динамической картинки "Пулково. Утренние рейсы".

Программа 8.15

```
'8Pulkovo.bas    Пулково. Утренние рейсы
10 CLS
20 SCREEN 9
30 COLOR 8, 15
40 DIM e(6)
50 LOCATE 1, 55
60 PRINT "Рейс  Время  Город"
'==1===== Блок рисования Санкт-Петербурга ==========================
70 CIRCLE (230, 20), 7, 8
80 DRAW "p14,8"
90 CIRCLE STEP(0, 0), 3, 8
100 DRAW "p8,8"
110 LOCATE 1, 22
120 PRINT "САНКТ-ПЕТЕРБУРГ"
'==2===== Блок расстановки городов ================================
130 DATA 20,245,19,2,"БЕЛГРАД", 100,300,23,9,"СОФИЯ"
140 DATA 500,300,23,60,"ТАШКЕНТ", 60,130,11,2,"ВАРШАВА"
150 DATA 560,170,14,68,"НОВОСИБИРСК", 230,275,21,27,"ОДЕССА"
160 FOR i = 1 TO 6
170 READ x(i), y(i), ky(i), kx(i), g$(i)
180 CIRCLE (x(i), y(i)), 5, 8
190 DRAW "p12,8"
200 CIRCLE STEP(0, 0), 2, 8
210 DRAW "p8,8"
220 LOCATE ky(i), kx(i)
230 PRINT g$(i)
240 NEXT i
'==3===== Блок начальных установок ================================
250 e(1) = 1
260 pov$(1) = "33":  x(1) = 253: y(1) = 50: yk(1) = 292: dx(1) = .98
270 pov$(2) = "-21": x(2) = 214: y(2) = 50: yk(2) = 292: dx(2) = -.47
280 pov$(3) = "0":   x(3) = 229: y(3) = 50: yk(3) = 267: dx(3) = 0
290 pov$(4) = "-33": x(4) = 202: y(4) = 50: yk(4) = 237: dx(4) = -.94
300 pov$(5) = "54":  x(5) = 269: y(5) = 43: yk(5) = 165: dx(5) = 2.3
310 pov$(6) = "-49": x(6) = 190: y(6) = 45: yk(6) = 124: dx(6) = -1.55
'==4===== Блок полета самолета ====================================
320 c1$ = "c15 nr2 h2 u5 m-10,-3 u2 r10 u4 m-4,-2 u1 bh2 c0 nr3 bf2"
330 c2$ = "c15 r14 d1 m-4,2 d4 r10 d2 m-10,+3 d5 g2 bu5 p15,15"
340 c$ = c1$ + c2$
400 DO
403 s = 0
   '----- Цикл рисования самолетов ---------------------------------
406 FOR k = 1 TO 6
409 IF e(k) = 1 THEN  ELSE 420
412 PSET (x(k), y(k)), 0
415 DRAW "ta" + pov$(k) + c$
```

```
420 IF y(k) > yk(k) THEN  e(k) = 0
421 s = s + e(k)                'вычисляет условие завершения программы
422 NEXT k
   '----- Блок анализа ------------------------------------------
423 GOSUB 500                                          'рисует трассы
424 IF s = 0 THEN EXIT DO                          'завершение программы
425 IF e2 = 0 THEN u1 = 1: e2 = 1: GOSUB 600             'сообщения
426 IF y(u + 1) > 90 AND u < 5 THEN  ELSE 430
427 u = u + 1: e(u + 1) = 1: GOSUB 600                   'сообщения
   '-----------------------------------------------------------------
430 FOR iv = 1 TO 20000: NEXT iv                      'выдержка времени
   '----- Цикл стирания самолетов ---------------------------------
431 FOR k = 1 TO 6
432 IF y(k) > yk(k) THEN e(k) = 0
433 IF e(k) = 1 THEN  ELSE 440
434 PSET (x(k), y(k)), 0
435 DRAW "ta" + pov$(k) + "bu2 p0,0"
440 NEXT k
   '----- Цикл приращения значения координат ----------------------
442 FOR k = 1 TO 6
444 IF e(k) = 1 THEN  ELSE 450
446 y(k) = y(k) + 1
448 x(k) = x(k) + dx(k)
450 NEXT k
   '-----------------------------------------------------------------
460 LOOP
470 GOTO 700
'==5===== Подпрограмма рисования трасс ============================
500 DATA 223,28,27,238, 220,27,68,124, 225,29,102,292
510 DATA 230,29,230,266, 234,29,493,293, 239,28,552,165
520 RESTORE 500
530 FOR j = 1 TO 6
540 READ x1(j), y1(j), x2(j), y2(j)
550 LINE (x1(j), y1(j))-(x2(j), y2(j)), 14
560 NEXT j
570 RETURN
'==6===== Подпрограмма вывода сообщений о рейсах ==================
600 u1 = u + 1
610 SELECT CASE u1
       CASE 1: nr$ = "5011  10:00  Ташкент"
       CASE 2: nr$ = "5012  10:20  София"
       CASE 3: nr$ = "5022  10:40  Одесса"
       CASE 4: nr$ = "5023  11:00  Белград"
       CASE 5: nr$ = "5035  11:20  Новосибирск"
       CASE 6: nr$ = "5036  11:40  Варшава"
620 END SELECT
630 LOCATE 1 + u1, 55: PRINT nr$
640 RETURN
'=================================================================
700 END
```

Результат работы программы 8.15 представлен на рис. 8.16.

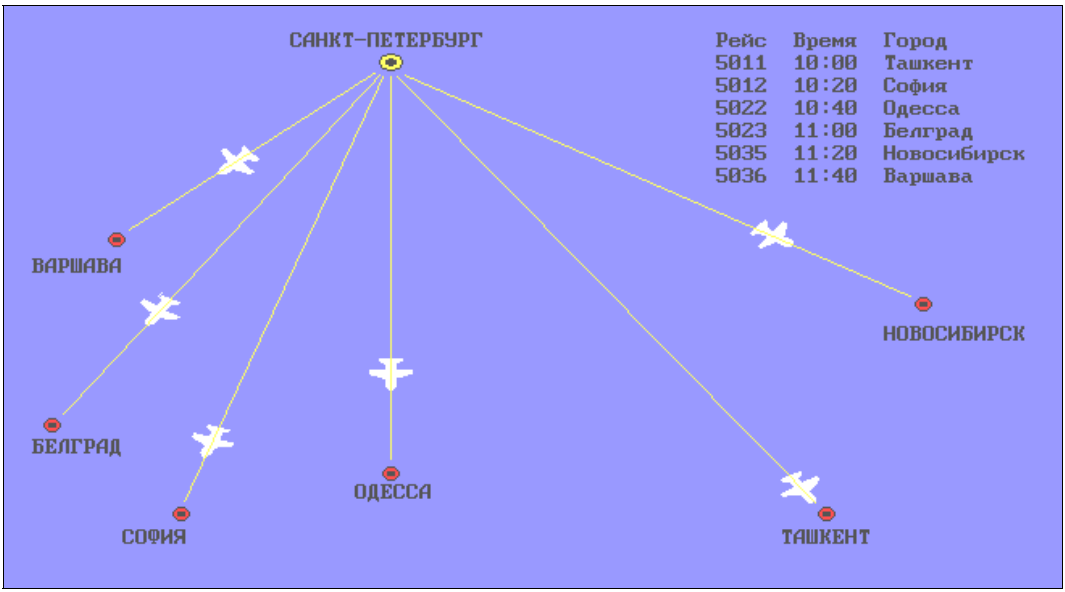


Рис. 8.16. Пулково. Утренние рейсы

Направление движения транспортного средства (в данном случае самолета) зависит от значений dx и dy. Его удобно определять с помощью диаграммы, приведенной на рис. 8.17. Отметим, что dx и dy не обязательно равны 1, как на диаграмме. Очевидно, что при dx = 1.4 и dy = 1.4 направление движения будет таким же, как и при dx = 1 и dy = 1, только увеличится скорость движения.

В программе 8.15 dy = 1 для всех самолетов, а вот значение dx определяется путем подбора. Сначала с помощью диаграммы на рис. 8.17 определяется ориентировочное значение dx, которое затем уточняется при запуске программы. Оно должно быть таким, чтобы самолет летел точно по соответствующей трассе. Для уточнения dx может понадобиться несколько прогонов программы.

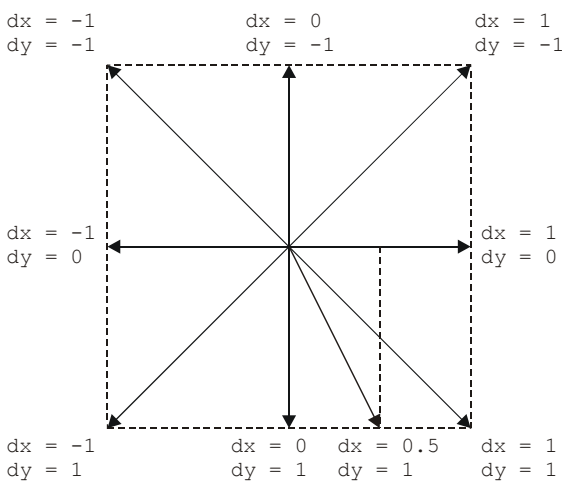


Рис. 8.17. Направление движения в зависимости от приращения координат

В процессе формирования траектории может возникнуть необходимость прерывания выполнения программы, что можно сделать, нажав комбинацию клавиш <Ctrl>+<Break>. Посмотреть результат на экране вывода можно с помощью клавиши <F4> (возврат — повторным нажатием клавиши <F4>). После использования комбинации <Ctrl>+<Break> запуск программы производится комбинацией клавиш <Shift>+<F5>, а не просто <F5>.

В блоке расстановки городов оператор `READ` в строке 170 считывает из операторов `DATA` в строках 130—150 значения координат `x(i)` и `y(i)` для рисования значка города операторами в строках 180—210, а также значения параметров `ky(i)` и `kx(i)` для операторов `LOCATE`, определяющих место вывода названия города оператором `PRINT` в строке 230.

Самолеты вылетают с интервалом в 90 единиц по переменной `y`, что зафиксировано в условии оператора `IF...THEN` в строке 426 блока анализа. После вылета переключателю `e` в строке 427 присваивается значение, равное 1. По завершению полета `k`-го самолета `e(k)` = 0. Если для всех самолетов соответствующие им `e(k)` = 0, то `s` в строке 421 будет равно 0, т. е. выполнится условие завершения программы, что зафиксируется с помощью условного оператора `IF...THEN` в строке 424 блока анализа.

Изображение самолета рисуется в строках 320—340 блока полета самолета. При выводе на экран оператором `DRAW` в строке 415 изображения самолетов будут отличаться углом поворота, вводимым в блоке начальных установок и задаваемым переменной `pov$(k)` в строке 415. Для вывода изображений самолетов, их стирания и задания приращения координат организованы циклы `FOR...NEXT` в строках 406—422, 431—440 и 442—450 соответственно.

Для такого количества самолетов подпрограмму рисования трасс можно было составить из шести операторов `LINE`, с непосредственно заданными значениями координат.

Двигаться по экрану может не только изображение, но и текст, например, в виде бегущей строки. На этом принципе основана программа 8.16 для соответствующего примера 8.14.

Пример 8.14. Бегущая строка

Требуется разработать программу рисования динамической картинки "В одно ухо влетело, из другого вылетело".

Программа 8.16

```
'8v-uho.bas    В одно ухо влетело...
10 CLS
20 SCREEN 9
30 COLOR 14
40 DIM x$(200)
50 PRINT : PRINT "  В ОДНО УХО ВЛЕТЕЛО, ИЗ ДРУГОГО ВЫЛЕТЕЛО"
'==1===== Блок рисования студентки ================================
60 CIRCLE (320, 104), 16, 12: PAINT STEP(0, 0), 12, 12       'голова
70 CIRCLE STEP(0, 24), 16, 13: PAINT STEP(0, 0), 13, 13      'блузка
80 LINE STEP(-16, 0)-STEP(32, 26), 13, BF                    'блузка
90 LINE -STEP(-32, 20), 5, BF                                'юбка
100 CIRCLE STEP(16, -58), 16, 12, , , .2: PAINT STEP(0, 0), 12, 12   'волосы
110 DRAW "c14 bm-14,+59 nr28 m+4,+49 r20 nm+4,-49 bh10 p14,14 bf10 "
                                                              'ноги
120 LINE -STEP(-20, 4), 6, BF                                'туфли
130 DRAW "f2 d2 r6 u2 e2 f2 d2 r6 u2 e2 bg3 p6,6 bl10 p6,6"   'туфли
```

```
'==2======== Блок ввода текста ====================================
140 FOR i = 1 TO 53
150 DATA Н,А,,З,А,Н,Я,Т,И,Я,,П,О,,И,Н,Ф,О,Р,М,А,Т,И,К,Е,,С,Л,Е,Д,У,Е,Т,,П
160 DATA Р,И,Х,О,Д,И,Т,Ь,,С,,Д,И,С,К,Е,Т,А,М,И
170 READ x$(i)
180 NEXT i
'==3======== Блок вывода бегущей строки ============================
200 FOR j = 1 TO 80
210 IF e = 0 THEN LOCATE 8, 44 ELSE LOCATE 8, 4
220 FOR i = j TO 34 + j
230 IF x$(i) = "" THEN PRINT SPC(1); x$(i);  ELSE PRINT x$(i);
240 NEXT i
250 IF a = 0 THEN SLEEP 1: a = 1                    'начальная выдержка
260 FOR kv = 1 TO 20000: NEXT kv                    'временная задержка
270 NEXT j
280 IF e = 0 THEN e = 2: GOTO 200           'возврат для вылета из уха
'==================================================================
600 END
```

Для ввода текста организован цикл `FOR...NEXT` в строках 140—180, в теле которого оператор `READ` считывает информацию из операторов `DATA` в строках 150—160. Две запятые рядом в операторе `DATA` означают пробел между словами, который выводится с помощью `SPC` при операторе `PRINT` в строке 230. Для вывода бегущей строки служит двойной (вложенный) цикл `FOR...NEXT` в строках 200—270. Внешний цикл отвечает за вывод текста, а внутренний — за сдвиг начала вывода. Отметим, что конечное значение счетчика цикла `FOR...NEXT` в строке 200, равное 80, заведомо больше количества букв фразы, чтобы при ее перезаписи стирались последние буквы.

Попробуйте самостоятельно, изменив программу 8.16, вдуть в ухо студентки какой-нибудь другой текст.

Результат работы программы 8.16 показан на рис. 8.18.

Теперь рассмотрим применение операторов `GET` и `PUT` для реализации движения.

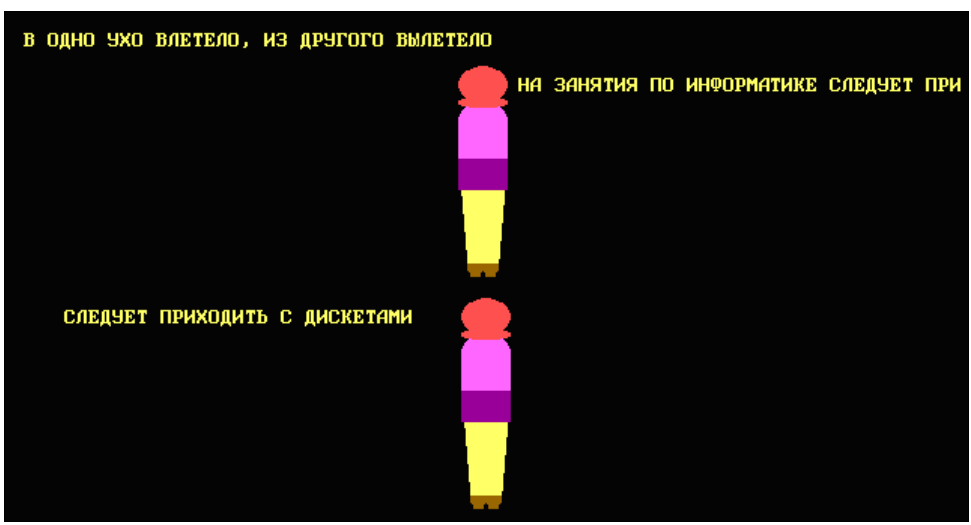


Рис. 8.18. В одно ухо влетело, из другого вылетело

Пример 8.15. Удар молнии

Требуется разработать программу рисования удара молнии.

Программа 8.17

```
'8molnia.bas    Молния
10 CLS
20 SCREEN 9
30 RANDOMIZE 32767 - TIMER
'==1===== Блок рисования молнии ====================================
40 DRAW "bm300,10 c14"
50 m$ = "nm-10,90 l1 nm-10,90 l1 m-10,90 r10 e1 nl9 e1 nl9"
60 DRAW m$ + m$ + m$ + "nm-6,+54 l1 nm-6,+54 l1 m-6,+54"
'==2===== Снятие изображения молнии ================================
70 B = INT((15 * 2 + 7) / 8 * 321)
80 DIM mol(B)
90 GET (287, 9)-STEP(14, 320), mol
100 FOR iv = 1 TO 50000: NEXT iv                    'выдержка времени
110 LINE (287, 9)-STEP(14, 320), 0, BF
'==3===== Блок удара молнии ========================================
120 DO UNTIL INKEY$ = CHR$(27)                      'клавиша Esc
130 SLEEP 2
140 n = INT(150 * RND(1))
150 PUT (300 + (-1) ^ n * n, 10), mol               'рисует молнию
160 FOR iv = 1 TO 50000: NEXT iv                    'выдержка времени
170 PUT (300 + (-1) ^ n * n, 10), mol               'стирает молнию
180 LOOP
'====================================================================
190 END
```

Нарисованная программой 8.17 молния показана на рис. 8.19.

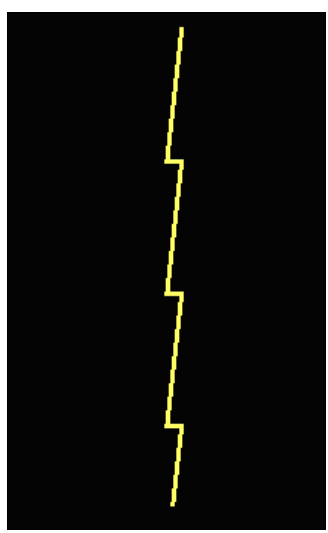


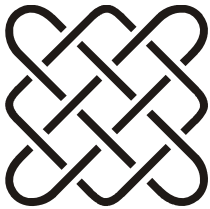
Рис. 8.19. Молния

Нарисованная операторами DRAW в строках 40—60 молния снимается в буфер оператором GET в строке 90. Для определения значений координат в операторе используется стирающий изображение оператор LINE в строке 110, приведенный к следующему виду:

```
LINE (287, 9)-STEP(14, 320), 10, B
```

в котором параметр BF заменен на B, а цвет фона изменен на другой, более заметный. Следует добиться, чтобы изображение полностью помещалось в рисуемую оператором LINE рамку, а затем перенести найденные значения координат в оператор GET. После этого оператору LINE придается прежний вид. Использование параметра STEP в операторах LINE и GET непосредственно дает значения x2 – x1 и y2 – y1 для формулы в строке 70, определяющей необходимый размер массива, в котором запоминается изображение.

Координата x начала рисования молнии определяется случайным образом, а величина ее смещения от 300 определяется в строке 140 посредством генератора случайных чисел RND. Оператор PUT в строке 150 выводит изображение молнии, а его повторное наложение оператором PUT в строке 170 — стирает. Выход из программы выполняется по нажатию клавиши <Esc>.



ГЛАВА 9

Работа с файлами

Файлом называется информация, хранимая на запоминающем устройстве (гибком или жестком диске). Каждый файл характеризуется именем, что позволяет выполнять поиск конкретного файла среди других файлов диска. В переводе с английского слово "file" означает "хранить информацию", а также "картотека", "досье". В информационных технологиях файл — это поименованная область памяти на внешнем носителе, предназначенная для хранения данных.

Имя файла в MS-DOS может содержать до 8 разрешенных символов. За именем после точки следует расширение файла (до трех символов), подсказывающее, какой программой создан соответствующий файл. Например, написанная на BASIC программа имеет расширение bas.

Если необходимые данные, полученные в процессе работы программы BASIC, требуют сохранения при ее последующих запусках, то целесообразно использовать файлы. Поясним это на примере.

Пример 9.1. Выбор максимального значения

Требуется создать программу, выбирающую случайным образом значение в диапазоне от 1 до 1000 и фиксирующую максимальное значение, полученное при всех запусках программы.

Программа 9.1

```
'9prim.bas
10 CLS
20 RANDOMIZE 32767 - TIMER
30 GOSUB 500                                  'считывает рекорд
40 PRINT : PRINT USING "   Рекорд ###"; r
'==1===== Блок получения результата ================================
50 y = (1 + 1000 * RND(1))                    'случайный выбор числа
60 IF y > r THEN x1 = y: GOSUB 600            'фиксация рекорда
70 PRINT
80 PRINT USING "   Результат ###"; y
90 GOTO 700
'==2===== Подпрограмма рекорда =====================================
500 OPEN "C:\BASIC\КНИГА\U1.BAS" FOR INPUT AS #2
510 INPUT #2, r                 'оператор, осуществляющий считывание
```

```
520 CLOSE #2
530 RETURN
   '----- Блок создания файла -------------------------------------
600 OPEN "C:\BASIC\КНИГА\U1.BAS" FOR OUTPUT AS #1
610 WRITE #1, x1
620 CLOSE #1
630 RETURN
'============================================================
700 END
```

Результат работы программы 9.1 показан на рис. 9.1.

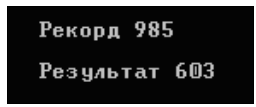


Рис. 9.1. Программа выбора значения

В строке 50 с помощью функции RND осуществляется выбор числа случайным образом (оператор RANDOMIZE 32767 – TIMER в строке 20 запускает генератор случайных чисел). Подпрограмма в строках 500—530 выводит предыдущее максимальное значение (рекорд), а подпрограмма в строках 600—630 записывает в файл новое значение рекорда при его появлении. Появление нового значения рекорда фиксируется в строке 60 с помощью условного оператора IF...THEN.

9.1. Операторы, управляющие работой файла

К операторам, управляющим работой файла, относится оператор OPEN, открывающий файл для создания, считывания или добавления, а также операторы, закрывающие файлы, например, оператор CLOSE.

OPEN — оператор ввода/вывода, включающий ввод/вывод в файл или устройство.

Например, следующий оператор открывает по 2-му каналу файл file.dat для создания. Если файл file.dat уже был создан, то он уничтожается и создается заново:

```
OPEN "file.dat" FOR OUTPUT AS #2
```

Здесь "file.dat" — символьное выражение, определяющее имя устройства или файла, включая путь;

OUTPUT — одно из описанных в табл. 9.1 ключевых слов, устанавливающих тип доступа к файлу;

#2 — номер файла, целое выражение от 1 до 255.

Таблица 9.1. *Типы доступа к файлу*

Тип	Комментарий
OUTPUT	Определяет последовательный вывод
INPUT	Определяет последовательный ввод
APPEND	Определяет последовательный вывод с добавлением, т. е. устанавливает указатель записи к концу файла. Вводимые данные к концу файла добавляют операторы PRINT # и WRITE #

Перед любыми операциями ввода/вывода, производимыми с файлом, его необходимо открыть. Оператор размещает буфер для ввода/вывода и устанавливает тип доступа к файлу.

Если тип опущен, то по умолчанию присваивается тип RANDOM, определяющий прямой ввод/вывод.

Следующий оператор открывает по 1-му каналу файл file2.dat для добавления:

```
OPEN "file2.dat" FOR APPEND AS #1
```

Если файл находится не в текущем каталоге, с которым работает BASIC, то в названии файла необходимо указывать полный путь к нему, например, "A:\ZZ\file.dat" для файла, находящегося на диске A: (дискете) в каталоге ZZ.

Если перед применением оператора OPEN с ключевым словом APPEND файл еще не существует, то он будет создан.

Следующий оператор открывает по 3-му каналу файл file2.dat для считывания:

```
OPEN "file2.dat" FOR INPUT AS #3
```

В конце оператора может стоять ключ LEN, задающий длину строки записи. Для файлов последовательного типа длина не означает длину каждой записи, т. к. такие файлы могут иметь записи различной длины. Она означает размер буфера, т. е. сколько символов загружаются в буфер перед записью или после чтения. Чем больше буфер, тем быстрее ввод/вывод. По умолчанию размер буфера равен 512 байт.

CLOSE — оператор ввода/вывода, закрывающий файл или устройство:

```
CLOSE #1, ... , #n, ...
```

где n — логический номер открытого файла.

Оператор CLOSE без аргументов закрывает все файлы и устройства.

CLOSE очищает весь буфер для закрываемого файла или устройства.

Например, следующий оператор закрывает файл, открытый по 1-му каналу:

```
CLOSE #1
```

RESET — оператор файлового ввода/вывода, автоматически закрывающий все файлы. Данный оператор закрывает все открытые файлы и записывает данные из буферов на диск. Все файлы, находящиеся на сменном носителе информации, обязательно должны быть закрыты перед удалением его из компьютера.

Операторы CLEAR, END, RUN и SYSTEM также автоматически закрывают все файлы.

EOF (от англ. "End Of File", код ASCII 26) — функция файлового ввода/вывода, определяющая условие конца файла:

```
EOF (n)
```

где n — номер открытого файла.

Функция возвращает единицу ("истина"), если указатель в файле последовательного доступа дошел до его конца. Эта функция может быть использована для тестирования конца файла при вводе данных, чтобы избежать появления сообщения об ошибке "Ввод после конца файла" ("Input past end of file").

9.2. Операторы, управляющие данными

К операторам, управляющим данными, будут относить операторы, записывающие данные в файл (например, PRINT #, PRINT # USING, WRITE #) и читающие данные из файла (INPUT #).

INPUT # — оператор файлового ввода/вывода, читающий данные из файла или последовательного устройства и присваивающий их переменным:

```
INPUT # n, x
```

где n — номер открытого файла, совпадающий с номером канала в операторе OPEN;

x — список переменных, которым присваиваются читаемые из файла значения. Типы данных должны соответствовать друг другу.

Ввод данных из файла производится так же, как ввод с клавиатуры. Данные могут разделяться пробелами, символами возврата каретки, перевода строки (ASCII 13 и 10) или запятыми. Начальные пробелы игнорируются. Знак конца файла <Ctrl>+<Z> (ASCII 26) завершает набор данных.

Если вводятся символьные строки, оператор игнорирует пробелы, а разделителями считаются запятые, символы конца строки и перевода каретки.

LINE INPUT # — оператор файлового ввода/вывода, читающий в указанную переменную символьную строку без разделителей из файла последовательного доступа:

```
INPUT # n, x$
```

где n — номер открытого последовательного файла;

x$ — символьная переменная, в которую считываются все символы текущей строки в файле до ее конца, т. е. до знаков ASCII 13 и 10.

INPUT$ — функция файлового ввода/вывода, читающая символьные строки из указанного файла:

```
INPUT (m, # n)
```

где n — номер открытого последовательного файла;

m — числовое выражение, задающее количество символов, читаемых из файла. Должно быть меньше или равно 32 767.

Если номер файла не указан, то символы читаются со стандартного устройства ввода (по умолчанию с клавиатуры). Знаки не дублируются на экран, т. е. ввод "без эха". Можно оборвать исполнение данной функции нажатием комбинации клавиш <Ctrl>+<Break>.

PRINT # и PRINT # USING — операторы файлового ввода/вывода, записывающие данные в последовательный файл.

PRINT # работает так же, как PRINT, и записывает данные в файл. Пробелы между элементами списка записываются в файл.

INPUT # — оператор, читающий данные из открытого по n-му каналу файла и присваивающий их значения переменной x$:

```
INPUT #n, x$
```

Здесь n — номер открытого файла последовательного типа; x$ — список выражений, содержащих записываемые в файл значения. Если он пуст, то записывается пустая строка.

WRITE # — оператор файлового ввода/вывода, посылающий данные в файл последовательного типа:

```
WRITE #m, x1$; x2$
```

где m — номер открытого оператором OPEN файла, который должен быть с ключевым словом OUTPUT или APPEND;

x1$, x2$ — список выражений, содержащий одно или несколько выражений, разделенных запятыми, значения которых вводятся в файл. Если список выражений опущен, то выводится пустая строка.

Например, следующий оператор в открытый по 2-му каналу файл записывает в одну строку из массива x10$ два символьных элемента с разделительными знаками:

```
WRITE #2, x10$ (1, j); x10$ (2, j)
```

Вводимые значения разделяются запятыми, символьные строки заключаются в кавычки. После печати последнего выражения в списке вставляются символы ASCII 13 (CR — возврат каретки) и ASCII 10 (LF — перевод строки). Оператор WRITE # записывает числовые значения без начальных и конечных пробелов.

9.3. Файлы последовательного типа доступа

Файлы последовательного доступа наиболее просты как в организации, так и в работе с ними. Записи обрабатываются последовательно одна за другой. Информация в таких файлах хранится в виде текста в кодах ASCII. Подобные файлы легко просмотреть на экране, используя любой простейший редактор, или в самом BASIC. Однако при большом информационном объеме файла обработка его заметно замедляется. В данном разделе будет рассматриваться работа с файлами последовательного доступа.

Для того чтобы начать работу с файлом, его следует создать. Рассмотрим программу, создающую файл.

Пример 9.2. Создание файла последовательного типа

Требуется разработать программу, создающую текстовый файл последовательного типа допуска.

Программа 9.2

```
'9OUTPUT.bas
10 CLS
20 f1$ = "63-я "
30 f2$ = "годовщина полного снятия блокады "
'==1===== Блок создания файла ====================================
40 OPEN "C:\BASIC\КНИГА\u2.bas" FOR OUTPUT AS #1
50 PRINT #1, f1$; f2$
60 CLOSE #1
'==================================================================
70 END
```

Оператор OPEN в строке 40 открывает файл u2.bas для создания по первому каналу, что показывает ключевое слово OUTPUT. Поскольку этот файл находится не в текущем каталоге, то в

его названии следует указывать и путь к нему. Полное название файла заключается в кавычки. Запись данных осуществляет оператор PRINT # в строке 50, а оператор CLOSE # в строке 60 закрывает созданный файл.

Теперь с созданным файлом можно производить и другие действия, например, добавлять данные.

Пример 9.3. Добавление данных в файл последовательного типа

Нужно разработать программу, добавляющую данные в текстовый файл последовательного типа допуска.

Программа 9.3

```
'9APPEND.bas
10 CLS
'==1===== Блок добавления в файл ===================================
20 f3$ = " 18 января 1944 года "
30 OPEN "C:\BASIC\КНИГА\u2.bas" FOR APPEND AS #3
40 PRINT #3, f3$
50 CLOSE #3
'====================================================================
60 END
```

Оператор OPEN в строке 30 открывает файл для добавления по третьему каналу, что показывает ключевое слово APPEND. Добавление производится с помощью оператора PRINT # в строке 40. После чего оператор CLOSE # в строке 50 закрывает файл.

Пример 9.4. Чтение данных из файла последовательного типа

Требуется разработать программу, считывающую данные из текстового файла последовательного типа допуска.

Программа 9.4

```
'9INPUT.bas
10 CLS
'==1===== Блок считывания из файла ================================
20 OPEN "C:\BASIC\КНИГА\u2.bas" FOR INPUT AS #2
30 INPUT #2, f1$, f2$
40 CLOSE #2
'====================================================================
50 PRINT f1$
60 PRINT f2$
70 END
```

Результат работы программы 9.4 дан на рис. 9.2.

Оператор OPEN в строке 20 открывает файл для считывания по второму каналу, что показывает ключевое слово INPUT. Данные записываются в строке 30 оператором INPUT # в переменные f1$, f2$ и выводятся на экран операторами PRINT в строках 50—60.

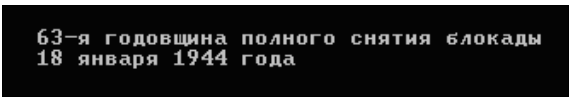


Рис. 9.2. Программа считывания данных из файла последовательного типа

При работе с файлами бывают случаи, когда необходимо использовать функцию EOF, определяющую ввод конца файла. Рассмотрим это на следующем примере.

Пример 9.5. Использование функции *EOF*

Требуется разработать программу, иллюстрирующую использование функции EOF.

Программа 9.5

```
'9EOF.bas
10 CLS
'==1===== Блок создания файла ======================================
20 OPEN "C:\BASIC\КНИГА\u4.bas" FOR OUTPUT AS #1
30 FOR i = 1 TO 8
40 DATA сон, нос, лом, мол, лот, тол, воз, зов
50 READ f1$(i)
60 PRINT #1, f1$(i)
70 NEXT i
80 CLOSE #1
'==2===== Блок чтения из файла =====================================
100 OPEN "C:\BASIC\КНИГА\u4.bas" FOR INPUT AS #2
110 DO
120 IF EOF(2) THEN EXIT DO
130 INPUT #2, f2$(k)
140 PRINT f2$(k)
150 LOOP
160 CLOSE #2
'====================================================================
200 END
```

Результат работы программы 9.5 приведен на рис. 9.3.

Рис. 9.3. Программа, иллюстрирующая действие функции EOF

В блоке создания файла (строки 20—80) в файл u4.bas последовательного типа доступа записываются 8 трехбуквенных слов. Оператор OPEN в строке 100 открывает файл для чтения. Для вывода данных оператором INPUT # в строке 130 организован цикл DO...LOOP в строках 110—150. Условие окончания цикла записано в строке 120 с помощью оператора IF...THEN,

что также позволяет избежать ошибки "Ввод после конца файла" ("Input past end of file"). Предлагаем читателю самостоятельно в этом убедиться, исключив из программы 9.5 строку 120.

9.4. Другие возможности

LOF — функция файлового ввода/вывода, возвращающая длину файла в байтах:

```
LOF(n)
```

где n — номер открытого файла.

При открытии файла любого типа доступа эта функция выдает его размер в байтах.

Пример 9.6. Использование функции *LOF*

Требуется разработать программу, демонстрирующую действие функции LOF.

Программа 9.6

```
'9LOF.bas
10 CLS
'==1===== Блок открытия файлов ====================================
20 OPEN "C:\BASIC\КНИГА\u1.bas" FOR INPUT AS #1
30 OPEN "C:\BASIC\КНИГА\u2.bas" FOR INPUT AS #2
40 OPEN "C:\BASIC\КНИГА\u3.bas" FOR INPUT AS #3
50 u1 = LOF(1)
60 u2 = LOF(2)
70 u3 = LOF(3)
80 CLOSE
'==2===== Блок вывода длины файлов ================================
90 PRINT "Длина файла u1.bas = "; u1
100 PRINT "Длина файла u2.bas = "; u2
110 PRINT "Длина файла u3.bas = "; u3
'==================================================================
120 END
```

Результат работы программы 9.6 приведен на рис. 9.4.

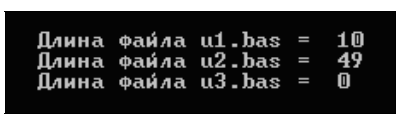


Рис. 9.4. Действие функции LOF

В строках 20—40 операторами OPEN открываются последовательно три файла для чтения (атрибут INPUT), длина которых в байтах определяется в строках 50—70, соответственно, с помощью функции LOF.

FILEATTR — функция файлового ввода/вывода, выдающая информацию об открытом файле:

```
FILEATTR(n, at)
```

где n — номер открытого файла;

at — атрибут, являющийся числовым выражением, которое имеет значение либо 1, либо 2. Этот атрибут определяет вид возвращаемой информации. Если атрибут равен 1, то возвращается тип файла. А если атрибут равен 2, то возвращается описатель файла DOS. Возвращаемые значения при атрибуте, равном 1, представлены в табл. 9.2.

Таблица 9.2. *Возвращаемые значения функцией FILEATTR при атрибуте, равном 1*

Тип доступа	Возвращаемое значение
INPUT	1
OUTPUT	2
RANDOM	4
APPEND	8
BINARI	32

Пример 9.7. Использование функции *FILEATTR*

Следующая программа иллюстрирует действие функции FILEATTR.

Программа 9.7

```
'9FILEATR.bas
10 CLS
20 OPEN "d.dat" FOR RANDOM AS #1
30 g = FILEATTR(1, 1)
40 IF g = 4 THEN f$ = "RANDOM"
50 PRINT "Тип доступа открытого файла — "; f$
60 END
```

Функция FILEATTR в строке 30 определяет тип (второй атрибут в скобках равен 1) открытого по 1-му каналу файла (номер файла или канала — первая цифра в скобках). Оператор IF...THEN в строке 40 переводит числовое значение в слово, выводимое на экран оператором PRINT в строке 50.

Результат работы программы 9.7 дан на рис. 9.5.

Рис. 9.5. Программа, определяющая тип доступа файла

FREEFILE — функция файлового ввода/вывода, выдающая следующий свободный номер файла. Эту функцию можно использовать, например, в подпрограммах-функциях или в подпрограммах-процедурах для того, чтобы они не использовали уже занятые номера файлов.

Пример 9.8. Использование функции *FREEFILE*

Рассмотрим программу, иллюстрирующую действие функции FREEFILE.

Программа 9.8

```
'9freefle.bas
10 CLS
20 INPUT "Задайте имя файла — ", f1$
30 f$ = "C:\BASIC\КНИГА\" + f1$ + ".bas"
40 GOSUB 200
50 PRINT "Файл "; f$; " открыт как #"; nf
60 INPUT "Будете открывать файл (Enter — да, 1 — нет) — ", v
70 PRINT
80 IF v = 1 THEN  ELSE 20
90 END
'==== Подпрограмма открытия файла ================================
200 nf = FREEFILE
210 OPEN f$ FOR OUTPUT AS nf
220 RETURN
'=================================================================
```

В строке 30 формируется полное имя файла, включающего путь и расширение, по сокращенному имени, введенному в строке 20 посредством оператора INPUT. Подпрограмма открытия для создания очередного файла вызывается в строке 40 оператором GOSUB.

Результат работы программы 9.8 показан на рис. 9.6.

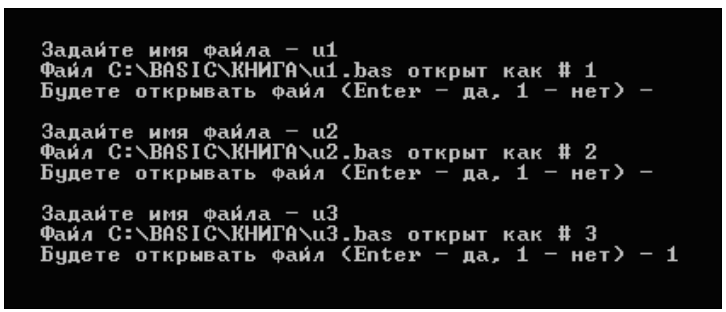


Рис. 9.6. Действие функции FREEFILE

LOC — функция файлового ввода/вывода, выдающая текущую позицию в файле:

```
LOC (n)
```

где n — номер открытого файла или устройства.

Для файлов последовательного типа доступа функция LOC возвращает текущую байтовую позицию, деленную на 128.

SEEK — оператор файлового ввода/вывода, устанавливающий позицию в файле для операций чтения или записи:

```
SEEK #n, m
```

где n — номер файла, открытого оператором OPEN;

m — числовое выражение, указывающее позицию для чтения или записи для файлов типа INPUT, OUTPUT, APPEND — номер байта от начала файла.

Позиция должна быть в пределах от 0 до 2 147 483 647 (что эквивалентно 2 ^ 31 – 1).

Выполнение оператора SEEK с отрицательным или нулевым значением позиции приведет к ошибке "Неверный номер записи" ("Bad record number").

Пример 9.9. Использование оператора *SEEK*

Следующая программа иллюстрирует действие оператора SEEK.

Программа 9.9

```
'9SEEK.bas
10 CLS
'==1===== Блок считывания из файла ================================
20 OPEN "C:\BASIC\КНИГА\u2.bas" FOR INPUT AS #2
30 SEEK #2, 6
40 INPUT #2, f1$, f2$
50 SEEK #2, 44
60 INPUT #2, f3$', f4$
70 CLOSE #2
'==2===== Блок вывода результата ==================================
80 PRINT f1$
90 PRINT f2$
100 PRINT
110 PRINT f3$
'===================================================================
120 END
```

Результат работы программы 9.9 показан на рис. 9.7.

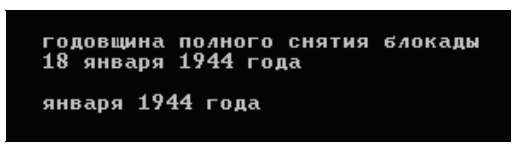
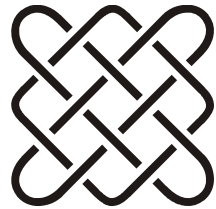


Рис. 9.7. Программа, демонстрирующая действие оператора SEEK. Чтение файла с 6 и 44 символов

Оператор OPEN в строке 20 открывает файл для чтения (атрибут INPUT). В строках 30, 50 операторы устанавливают позицию начала считывания 6 и 44 соответственно. Посредством операторов PRINT результат выводится на экран.



ГЛАВА 10

Работа со строковыми переменными

Для работы с текстом в BASIC имеются строковые константы и переменные. В некоторых источниках их именуют также символьными или текстовыми. Объявляются строковые переменные с помощью служебного слова `STRING` или путем добавления к имени суффикса `$` — знак доллара. При инициализации в них записывается и далее хранится пустая строка или нуль-строка (строка, в которой нет никаких символов) до тех пор, пока переменным не будет присвоено какое-либо иное значение. Если же строковая переменная объявлена следующим образом:

```
DIM cl AS STRING * 3
```

то до использования она хранит 3 пробела. Такое объявление задает максимальную длину переменной — 3 символа.

В строковых переменных могут содержаться как отдельные символы, так и их последовательности длиной до 255 символов. К ним в BASIC могут применяться специальные операции, о которых речь пойдет в данной главе.

Каждый символ, представленный на клавиатуре для компьютера, переводится в числовой код. Эти коды объединены в стандартную международную таблицу кодов ASCII (American Standard Code for Information Interchange). Коды с 0 по 32 не имеют изображения на экране и служат для функций управления (клавиши управления курсором, пробел и пр.). Далее следуют знаки препинания, цифры, строчные и прописные буквы латинского алфавита и другие символы, находящиеся на клавиатуре. Всего их 128. Последующие 128 кодов в диапазоне от 129 до 255 служат для расширения возможностей клавиатуры, например для генерации национальных символов — в нашем случае для кириллицы. Для работы с этими кодами в BASIC предусмотрены функции `ASC` (от ASCII) и `CHR$`.

10.1. Функции и операторы обработки символьных строк

10.1.1. Функции *CHR$* и *ASC*

`CHR$` — символьная функция, возвращающая строку из одного символа, ASCII-код которого является ее аргументом:

```
CHR$(n)
```

где n — числовое выражение, один из кодов ASCII в диапазоне от 0 до 255 (в противном случае интерпретатор сообщит об ошибке "Ошибочный аргумент функции" ("Illegal function call"). Этот целочисленный аргумент имеет тип INTEGER.

Пример 10.1. Использование функции *CHR$*

Следующая программа реализует известный сюжет про глаза, мерцающие в темноте, и мысли Штирлица по этому поводу.

Программа 10.1

```
'10CHR$.bas    Глаза в темноте
10 CLS
20 SCREEN 9
30 COLOR 14
40 RANDOMIZE 32767 - TIMER
'==1===== Блок движения глаз ========================================
50 DO UNTIL INKEY$ = CHR$(27)                          'клавиша Esc
60 dx = INT(50 * RND(1))
70 dy = INT(50 * RND(1))
80 x = 250 + dx
90 y = 60 + dy
100 GOSUB 200
110 SLEEP 2
120 LINE (0, 40)-STEP(600, 200), 0, BF
130 SLEEP 1
140 c = c + 1
150 IF c = 5 THEN PRINT "- Дятел"; CHR$(44); " - подумал Штирлиц."
160 IF c = 8 THEN  ELSE 180
170 PRINT "- Сам ты - дятел"; CHR$(44); " - подумал Мюллер."
180 LOOP
190 GOTO 250
'==2===== Подпрограмма рисования глаз ============================
200 CIRCLE (x, y), 8, 2, , , .4: PAINT STEP(0, 0), 10, 2
210 CIRCLE STEP(30, 0), 8, 2, , , .4: PAINT STEP(0, 0), 10, 2
220 LINE STEP(-3, -2)-STEP(6, 4), 4, BF
230 LINE STEP(-36, -4)-STEP(6, 4), 4, BF
240 RETURN
'=================================================================
250 END
```

В приведенной программе функция CHR$ используется в двух случаях. В строке 50 она используется для формирования условия выхода из цикла, а в строках 150 и 170 — для вывода запятой с помощью оператора PRINT (запятая в операторе PRINT является разделительным знаком, и ее нельзя просто поставить в кавычки).

Для реализации мерцания и небольшого перемещения пары глаз в программе 10.1 организован цикл DO...LOOP в строках 50—180. Приращение к величине координат опорной точки (x = 250 и y = 60) определяется случайным образом в строках 60—70. Оператор GOSUB 200 в строке 100 обеспечивает вывод на экран горящих глаз, а оператор LINE в строке 120 стирает

их. Операторы SLEEP в строках 110 и 130 задают, соответственно, время, когда глаза горят и когда они погашены.

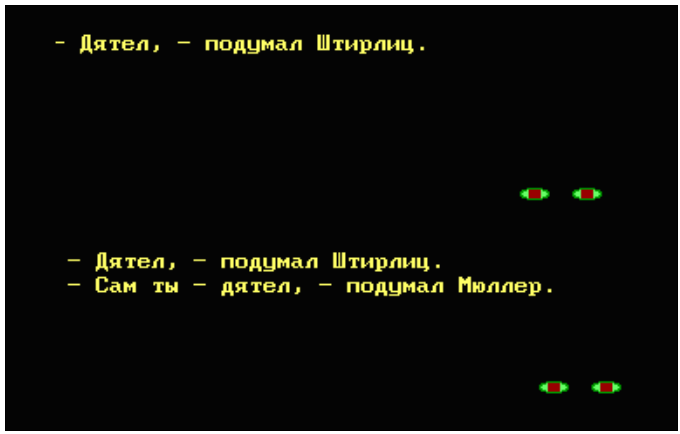


Рис. 10.1. Результат работы программы про глаза, мерцающие в темноте

Результат работы программы 10.1 приведен на рис. 10.1.

Обратной функции CHR$ является функция ASC, возвращающая числовое значение ASCII-кода первого символа в символьном выражении:

```
ASC(sv$)
```

где sv$ — символьное выражение.

Пример 10.2. Использование функции *ASC*

Следующая программа иллюстрирует работу функции ASC.

Программа 10.2

```
'10ASC.bas
10 CLS
20 k1 = ASC("Y")
30 PRINT " Код заглавной буквы Y = "; k1
40 k2 = ASC("FOCKO")
50 PRINT " Код первой буквы слова FOCKO = "; k2
60 END
```

Результат работы программы 10.2 приведен на рис. 10.2.

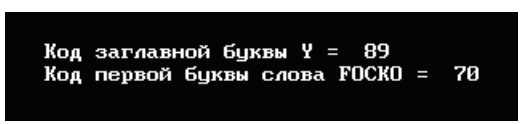


Рис. 10.2. Действие функции ASC

10.1.2. Функция *LEN*

LEN — символьная функция, возвращающая количество символов данной строки или количество байт, занимаемых переменной:

```
k = LEN(g$)
k = LEN(p)
```

где g$ — символьное выражение (строка символов);

p — переменная, объект любого типа, для которого требуется узнать занимаемый объем памяти.

Поскольку по умолчанию переменная a — обычной точности, то:

```
PRINT LEN(a)
```

выведет на экран 4 — занимаемый объем в памяти переменными данного типа в байтах.

Пример 10.3. Использование функции *LEN*

Программа, иллюстрирующая работу оператора LEN, определяет, кто из двоих загадал на заданную букву слово длиннее.

Программа 10.3

```
'10LEN.bas
10  CLS
20  PRINT : b$ = "На какую букву слово — "
30  PRINT b$;
40  INPUT "", s$
50  PRINT
60  INPUT "1-е слово — ", p$
70  s1 = LEN(p$)
80  CLS : PRINT b$ + s$
90  PRINT
100 LINE INPUT ; "2-е слово — ", o$
110 s2 = LEN(o$)
120 PRINT "  число букв — "; s2
130 PRINT "1-е слово — "; p$; "  число букв — "; s1
140 IF s1 = s2 THEN PRINT : PRINT "ничья": GOTO 170
150 IF s1 > s2 THEN ig = 1 ELSE ig = 2
160 PRINT : PRINT "Победил"; ig; "-й игрок"
170 END
```

Результат работы программы 10.3 приведен на рис. 10.3.

Оператор CLS в строке 80 стирает придуманное первым игроком слово перед тем, как второй игрок будет вводить свое слово. Операторы PRINT в строках 20, 30, 50, 90, 120—140, 160 обеспечивают интерфейс программы. Операторы в строках 40, 60, 100 служат для ввода, соответственно, буквы, на которую начинаются слова, слова 1-го игрока и слова 2-го игрока. Причем в строке 100 использован оператор LINE INPUT с точкой с запятой перед приглашением к вводу для того, чтобы выводимое оператором PRINT в строке 120 число букв печаталось в той же строке. Сразу же после ввода слова с помощью LEN в строках 70 и 110 определяется длина придуманных слов, которая сравнивается в строках 140—150.

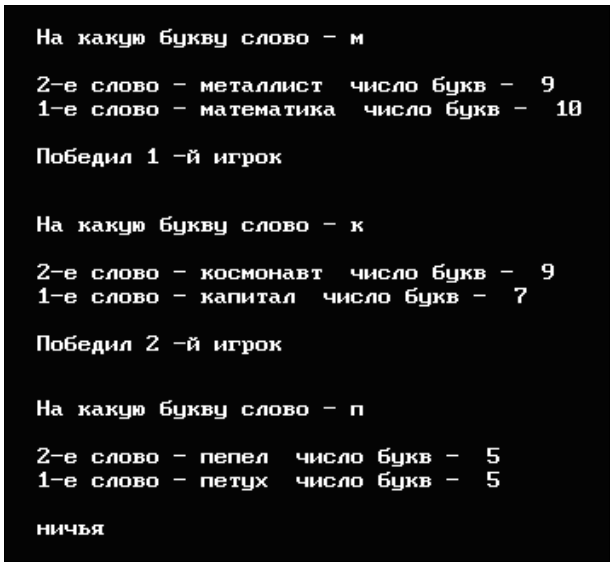


Рис. 10.3. Программа-игра: чье слово длиннее

10.1.3. Функции *STRING$* и *SPACE$*

STRING$ — символьная функция, возвращающая строку, заполненную символами данного ASCII-кода или данным символом:

```
STRING$(p, m)
STRING$(p, m$)
```

где p — числовое выражение, длина строки;

m — числовое выражение в пределах от 0 до 255, ASCII-код символа-заполнителя строки;

m$ — символьное выражение, указывающее на строку, первый символ которой используется для заполнения строки.

Пример 10.4. Использование функции *STRING$*

Программа, иллюстрирующая работу символьной функции STRING$, рисует флаг России с помощью псевдографики.

Программа 10.4

```
'10flag.bas
10 CLS
20 FOR i = 1 TO 15
30 IF i <= 5 THEN
   cv = 15
   ELSEIF 5 < i AND i < 11 THEN cv = 9
   ELSEIF i > 10 THEN cv = 12
40 END IF
50 COLOR cv
60 LOCATE 3 + i, 10
```

```
70 PRINT STRING$(60, "Ш")
80 NEXT i
90 END
```

Флаг рисуется в пятнадцати строках, по пять строк на каждый цвет. Для изменения цвета в операторе COLOR в строке 50 используется условный оператор IF...THEN в блочной форме (строка 30). Оператор STRING$ в строке 70 выводит строки длиной 60, заполненные русской заглавной буквой "Ш". Результат работы программы 10.4 дан на рис. 10.4.

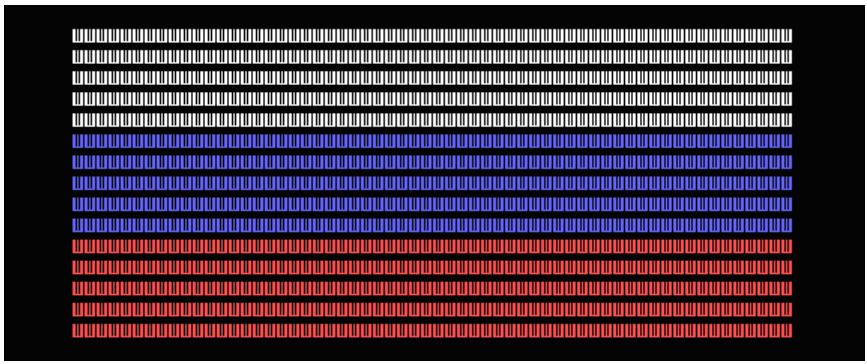


Рис. 10.4. Псевдографика. Флаг России

SPACE$ — символьная функция, возвращающая строку пробелов заданной длины:

```
SPACE$(n)
```

где n — числовое выражение в пределах от 0 до 32 767, определяющее количество пробелов в строке.

Пример 10.5. Использование функции *SPACE$*

Следующая программа иллюстрирует работу символьной функции SPACE$.

Программа 10.5

```
'10SPACE$.BAS
10 CLS
20 FOR k = 1 TO 4
30 PRINT SPACE$(k); k
40 NEXT k
50 END
```

Результат работы программы 10.5 представлен на рис. 10.5.

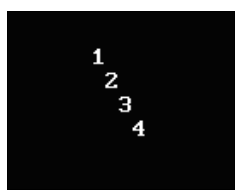


Рис. 10.5. Действие функции SPACE$

10.1.4. Функции *STR$* и *VAL*

STR$ — символьная функция, возвращающая символьное представление числа или числового выражения:

```
STR$(n)
```

где n — число или числовое выражение.

Если числовое выражение положительное, то строка возвращается с начальным пробелом. Превращение числа в символьное выражение позволяет производить над ним те же действия, что и с символьной строкой: складывать, выделять фрагменты числа, заменять, форматировать и пр. Например, можно использовать циклы применительно к графическому оператору DRAW.

Пример 10.6. Использование функции *STR$*

Требуется разработать программу, выводящую 6 квадратов в линию.

Программа 10.6

```
'10STR$.bas
10 CLS
20 SCREEN 9
30 FOR x = 100 TO 300 STEP 40
40 x$ = STR$(x)
50 DRAW "bm" + x$ + ",100 c14 u20 r20 d20 l20"
60 NEXT x
70 END
```

Результат работы программы 10.6 показан на рис. 10.6.

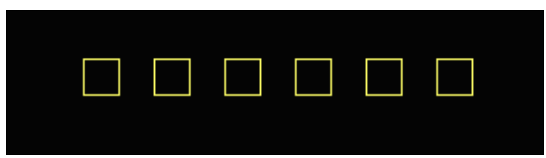


Рис. 10.6. Квадраты

В процессе работы цикла FOR...NEXT в строках 30—60 на первом шаге оператор DRAW в строке 50 будет выглядеть следующим образом:

```
DRAW "bm100,100 c14 u20 r20 d20 l20"
```

На втором шаге:

```
DRAW "bm140,100 c14 u20 r20 d20 l20"
```

На третьем шаге:

```
DRAW "bm180,100 c14 u20 r20 d20 l20"
```

и т. д.

Причем значения аргумента функция STR$(x) в строке 40 берет из счетчика цикла.

Обратной функции STR$ является функция VAL, переводящая строковое представление числа в число:

```
VAL(n$)
```

где n$ — строковое представление числа.

Пример 10.7. Использование функции *VAL*

Требуется разработать программу, иллюстрирующую работу функции VAL.

Программа 10.7

```
'10VAL.BAS
10 CLS
20 n$ = DATE$
30 g1 = VAL(RIGHT$(n$, 4))
40 INPUT "Укажите год Вашего рождения – ", g2
50 PRINT "Вы прожили"; g1 – g2; "лет"
60 END
```

Функция DATE$ выводит текущую дату в символьном виде. Функция RIGHT$ в строке 30 вырезает из даты четыре символа справа, т. е. год в символьном виде. Оператор VAL преобразует год в число, годное для использования в арифметических операциях, что и реализуется в строке 50.

Результат работы программы 10.7 дан на рис. 10.7.

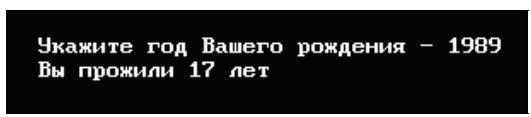


Рис. 10.7. Иллюстрация работы функции VAL

10.1.5. Функции *RIGHT$* и *LEFT$*

RIGHT$ — символьная функция, возвращающая k правых символов символьной строки:

```
RIGHT$(n$, k)
```

где n$ — исходная строка, символьное выражение;

k — числовое выражение в пределах от 0 до 32 767, указывающее, сколько берется символов строки справа. Если k = 0, то возвращается пустая строка.

LEFT$ — символьная функция, возвращающая строку, содержащую k левых символов исходной строки:

```
LEFT$(n$, k)
```

где n$ — исходная строка, символьное выражение;

k — числовое выражение в пределах от 0 до 32 767. Если k = 0, то возвращается пустая строка.

Пример 10.8. Сокращение слов

Требуется разработать программу, производящую сокращение исходного слова.

Программа 10.8

```
'10LEFT$.bas
10 CLS : SCREEN 9
20 n$ = "функция"
30 l$ = LEFT$(n$, 1)
40 p$ = RIGHT$(n$, 3)
50 s$ = l$ + "-" + p$
60 PRINT "Слово  "; n$; "  в сокращенной форме — "; s$
70 END
```

В сокращенной форме дефис соединяет один символ слева, вырезанный из исходного слова функцией `LEFT$`, и три символа справа, вырезанных функцией `RIGHT$`.

Результат работы программы 10.8 приведен на рис. 10.8.

Рис. 10.8. Программа сокращения слова

10.1.6. Функция и оператор *MID$*

`MID$` — символьная функция, возвращающая фрагмент исходной строки:

```
MID$(n$, k, m)
```

где `n$` — исходная строка, символьное выражение;

`k` — числовое выражение в пределах от 1 до 32 767, определяющее позицию начального символа фрагмента исходной строки;

`m` — числовое выражение, задающее длину фрагмента.

Если длина фрагмента опущена или количество символов от начала до конца строки меньше длины, то функция `MID$` возвращает фрагмент от начала до конца строки. Если значение позиции начального символа фрагмента больше, чем длина исходной строки, то функция `MID$` возвращает пустую строку.

Пример 10.9. Использование функции *MID$*

Требуется разработать программу, производящую грамматический разбор слова.

Программа 10.9

```
'10MID$-F.bas
10 CLS
'==1===== Блок ввода слова ======================================
20 PRINT "Грамматический разбор слова — ";
30 f$ = "подводный"
```

```
40 PRINT f$
50 PRINT
'==2===== Блок разбора слова ======================================
60 FOR i = 1 TO 4
70 DATA "Приставка — ",1,3, "Корень — ",4,3
80 DATA "Суффикс — ",7,1, "Окончание — ",8,2
90 READ a$, n, m
100 p$ = MID$(f$, n, m)
110 PRINT a$; p$
120 NEXT i
'==================================================================
130 END
```

Результат работы программы 10.9 дан на рис. 10.9.

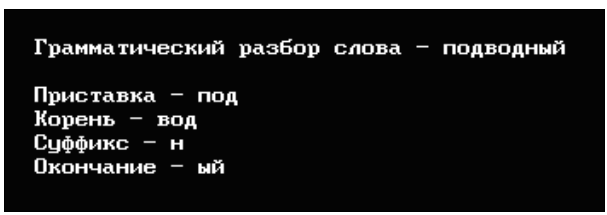


Рис. 10.9. Программа с функцией MID$

Значения аргументов n, m для функции MID$ в строке 100 вводятся с помощью пары операторов READ/DATA в строках 70—90. Указанные операторы вместе с выводящим результаты работы программы оператором PRINT в строке 110 составляют тело цикла FOR...NEXT в строках 60—120.

Предлагаем читателю самостоятельно разработать программу, в которой разбираемое слово и аргументы для функции MID$ задаются в диалоговом режиме с помощью оператора INPUT.

MID$ — символьный оператор, заменяющий фрагмент строки на содержимое другой строки:

```
MID$(n$, k, m) = v$
```

где n$ — символьная переменная, в которой требуется произвести изменения;

k — числовое выражение, определяющее начальную позицию в исходной строке, с которой производится замена;

m — количество символов из символьного выражения, которые следует заменить;

v$ — символьное выражение, представляющее новые символы, на которые следует заменить.

Аргументы k и m — целые выражения. Аргумент n$ должен быть только символьной переменной, а символьное выражение v$ может быть переменной, константой или выражением.

Если длина m опущена, то производится замена полным содержанием символьного выражения v$. Если же длина m задана, то производится замена указанным количеством символов слева из символьного выражения v$. Заменяемый фрагмент не может превысить текущей длины переменной.

Пример 10.10. Использование оператора *MID$*

Следующая программа иллюстрирует работу оператора MID$.

Программа 10.10

```
'10MID$-O.bas
10 CLS
20 f$ = "Лондон, Франция..."
30 PRINT f$
40 PRINT "Неверно — заменить"
50 PRINT
60 PRINT "Длина задана m = 12"
70 MID$(f$, 9, 12) = "Великобритания"
80 PRINT f$
90 PRINT "Длина задана m = 6"
100 MID$(f$, 9, 6) = "Англия......."
110 PRINT f$
120 PRINT "Длина не задана"
130 MID$(f$, 9) = "Англия......."
140 PRINT f$
150 END
```

В операторе MID$ в строке 70 задана длина заменяемого фрагмента, равная 12. Но, как видно из рис. 10.10, было заменено только 10 символов ("Великобрит"), поскольку в исходной символьной строке от 9 символа до конца строки находится 10 символов. В операторе в строке 100 определено, что количество заменяемых символов — 6, что и было произведено. В операторе в строке 130 длина строки не задана, поэтому замена была произведена из заменяющего символьного выражения до конца строки, что и видно из рис. 10.10.

Результат работы программы 10.10 показан на рис. 10.10.

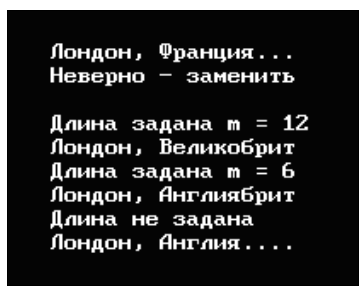


Рис. 10.10. Программа, показывающая работу оператора MID$

10.1.7. Функция *INSTR*

INSTR — символьная функция, возвращающая позицию вхождения подстроки в указанной строке:

```
INSTR(p, c1, c2)
```

где p — начало, т. е. условное смещение от начала строки, устанавливающее позицию начала поиска. Число p должно быть в пределах от 1 до 32 767. Если p не задано, то функция начинает поиск с первого символа строки c1;

c1 — строка, в которой производится поиск;

c2 — искомая подстрока.

Значение, возвращаемое функцией, зависит от следующих условий, приведенных в табл. 10.1.

Таблица 10.1. *Значения, возвращаемые функцией* INSTR

Условие	Значение
Строка c2 найдена в строке c1	Позиция p, с которой строка c2 начинается в строке c1
Начало p больше длины строки c1	0
Строка c1 пустая	0
Строка c2 не найдена	0
Строка c2 пустая	Начало (если есть), иначе 1

Пример 10.11. Использование функции *INSTR*

Программа, иллюстрирующая работу функции INSTR, проверяет вхождение заданного фрагмента в имеющиеся тексты.

Программа 10.11

```
'10INSTR.bas
10 CLS
20 INPUT "Что хотите найти — ", f$
30 t$(1) = "Бейсик — это язык программирования высокого уровня"
40 t$(2) = "для ввода информации с клавиатуры служит оператор INPUT"
50 t$(3) = "оператор PRINT выводит информацию на экран"
60 FOR i = 1 TO 3
70 n = INSTR(1, t$(i), f$)
80 IF n > 0 THEN
     PRINT "Найдено"
     PRINT t$(i)
     a = 1
90 END IF
100 NEXT i
110 IF a = 0 THEN PRINT "Не найдено"
120 END
```

Посредством оператора INPUT в строке 20 вводится ключевое слово для поиска. Затем с помощью функции INSTR в строке 70, являющейся телом цикла FOR...NEXT в строках 60—100, проверяется вхождение заданного слова в имеющиеся тексты в строках 30—50. Если искомое слово входит в какой-либо текст, то, как видно из табл. 10.1, функция INSTR возвращает значение больше 1. Результаты поиска обрабатываются условным оператором в блочной

форме IF...THEN в строках 80—90. Внимание, вместо русской буквы "р" следует вводить латинское "p", т. к. QBASIC "не выговаривает" эту букву.

Результаты поиска, осуществленные программой 10.11, можно увидеть на рис. 10.11.

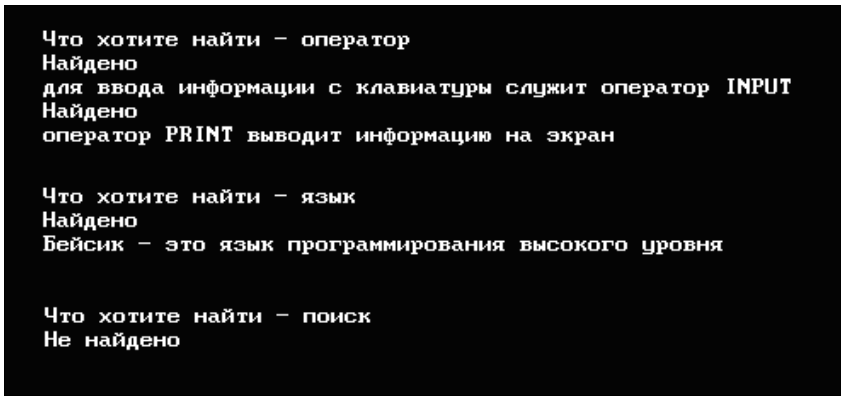


Рис. 10.11. Программа, иллюстрирующая работу оператора INSTR

10.1.8. Функции *HEX$* и *OCT$*

HEX$ — символьная функция, возвращающая шестнадцатеричное представление десятичного аргумента:

```
HEX$(n)
```

где n — любое выражение в десятичном виде.

OCT$ — символьная функция, возвращающая символьное восьмеричное представление десятичного аргумента:

```
OCT$(n)
```

где n — числовое выражение, может быть любого типа.

Пример 10.12. Использование функций *HEX$* и *OCT$*

Следующая программа иллюстрирует работу функций HEX$ и OCT$.

Программа 10.12

```
'10HEX.bas
10 CLS
20 INPUT "Задайте десятичное число — ", f
30 n$ = HEX$(f)
40 PRINT "Шестнадцатеричное представление = "; n$
50 m$ = OCT$(f)
60 PRINT "Восьмеричное представление = "; m$
70 END
```

Результат работы программы 10.12 показан на рис. 10.12.

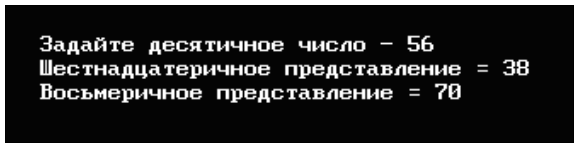


Рис. 10.12. Работа функций HEX$ и OCT$

10.2. Строковые операции

Над строками можно производить следующие действия: конкатенацию и сравнение строк.

Конкатенация — это сложение двух символьных строк. Для конкатенации используется знак плюс (+).

Пример 10.13. Конкатенация строк

Требуется разработать программу, осуществляющую конкатенацию двух символьных строк.

Программа 10.13

```
'10konkat.bas
10 CLS
20 s1$ = "супер"
30 s2$ = "маркет"
40 s$ = s1$ + s2$
50 PRINT "1-я строка — "; s1$
60 PRINT "2-я строка — "; s2$
70 PRINT "Конкатенация — "; s$
80 END
```

Результат работы программы 10.13 приведен на рис. 10.13.

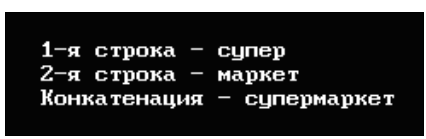


Рис. 10.13. Пример конкатенации

Сравнение строк производится при помощи операторов сравнения:

<>, =, <, >, <=, >=

Сравнение строк производится в соответствии с ASCII-кодами каждого символа в сравниваемых строках. Если ASCII-коды равны, то строки считаются равными. Строка, символы которой имеют большие ASCII-коды, считается большей. Если одна из сравниваемых строк короче другой, а до этого места строки были равными, то она считается меньшей. При сравнении учитываются начальные и конечные пробелы. Если строки содержат числа, то сравнение этих строк можно использовать для оценки их содержимого. Далее приведены примеры правильных операций сравнения:

```
"aa" < "bb"
"супермаркет" = "супер" + "маркет"
"пол" < "полы"
"КМ" < "км"
"kk " > "kk"
"x&" > "x#"
"05-06-06" < "07-06-06"
```

Пример 10.14. Сортировка по алфавиту

Требуется разработать программу, осуществляющую сортировку слов по алфавиту путем сравнения строк.

Программа 10.14

```
'10Sort.bas    Программа сортировки слов по алфавиту
10 CLS
20 DIM g$(10)
30 LOCATE 2, 5
40 PRINT "Исходный список"
50 LOCATE 2, 27
60 PRINT "Отсортированный"
'==1===== Блок ввода и вывода исходного списка ======================
70  DATA Тверь,Москва,Вологда,Петербург,Омск,Курск,Казань
80  DATA Sofia,Berlin,Rim
90  FOR i = 1 TO 10
100  READ g$(i)
110 LOCATE 2 + i, 10
120 PRINT g$(i)
130 NEXT i
'==2===== Блок сортировки ============================================
140 FOR i = 1 TO 9
150 FOR j = i + 1 TO 10
160 IF g$(i) > g$(j) THEN SWAP g$(i), g$(j)
170 NEXT j, i
'==3===== Блок вывода результатов сортировки =======================
180 FOR i = 1 TO 10
190 LOCATE 2 + i, 30
200 PRINT g$(i)
210 NEXT i
'====================================================================
220 END
```

Результат работы программы 10.14 дан на рис. 10.14.

Ввод списка городов осуществляется операторами `READ/DATA` в строках 70—80, 100, а вывод — с помощью операторов `LOCATE` и `PRINT` в строках 110—120 (исходный список) и в строках 190—200 (отсортированный). Центральным в программе является сравнение элементов исследуемой последовательности городов посредством оператора отношения в строке 160. Для проведения сортировки организован двойной цикл `FOR...NEXT` в строках 140—170. На первом шаге цикла по `i` выбирается наименьший элемент массива и ставится с помощью оператора `SWAP` на первое место. На втором шаге выбирается наименьший из

оставшихся элементов и ставится на второе место. Продолжая эту процедуру, получаем отсортированный список.

Рис. 10.14. Программа сортировки по алфавиту

10.3. Другие возможности

В этот раздел авторы отнесли возможности BASIC, которые им на практике применять не приходилось. Возможно, читателю они когда-нибудь и пригодятся.

10.3.1. Функции *LTRIM$* и *RTRIM$*

`LTRIM$` — символьная функция, возвращающая копию строки с удаленными начальными пробелами:

```
LTRIM$(f$)
```

где `f$` — символьное выражение с начальными пробелами.

`RTRIM$` — символьная функция, возвращающая символьную строку с удаленными правыми пробелами:

```
RTRIM$(f$)
```

где `f$` — символьное выражение с конечными пробелами.

Пример 10.15. Использование функций *LTRIM$* и *RTRIM$*

Следующая программа иллюстрирует работу функций `LTRIM$` и `RTRIM$`.

Программа 10.15

```
'10RLTRIM.bas
10 CLS
'==1===== Блок формирования слова ==================================
20 f1$ = "гидро   "                 'фрагмент с конечными пробелами
30 f2$ = "электро"                  'фрагмент без пробелов
40 f3$ = "   станция"               'фрагмент с начальными пробелами
50 PRINT "Вывод с пробелами"
60 PRINT f1$ + f2$ + f3$
```

```
'==2===== Блок удаления пробелов ==================================
70  f5$ = LTRIM$(f3$)               'удаляет начальные пробелы
80  f6$ = RTRIM$(f1$)               'удаляет конечные пробелы
90  PRINT
100 PRINT "Вывод с удаленными пробелами"
110 PRINT f6$ + f2$ + f5$
'=================================================================
120 END
```

Результат работы программы 10.15 показан на рис. 10.15.

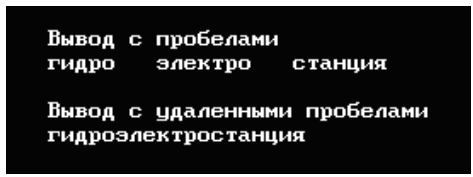


Рис. 10.15. Программа, удаляющая пробелы

10.3.2. Функции *LCASE$* и *UCASE$*

`LCASE$` — символьная функция, возвращающая символьную строку, в которой все латинские буквы преобразованы в строчные:

```
LCASE$(f$)
```

где `f$` — символьная строка, может быть переменной или фиксированной длины.

Данная функция не преобразует символы кириллицы.

`UCASE$` — символьная функция, возвращающая символьное значение, в котором все латинские буквы — заглавные:

```
UCASE$(f$)
```

где `f$` — символьная строка.

Данная функция не преобразует символы кириллицы. Функция `UCASE$` может работать со строками переменной или фиксированной длины.

Пример 10.16. Управление регистром

Следующая программа иллюстрирует работу операторов `LCASE$` и `UCASE$`.

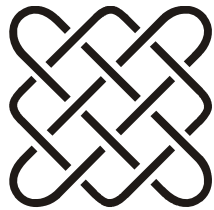
Программа 10.16

```
'10LUCASE.bas    По когтям узнают льва
10 CLS
'==1===== Блок вывода исходного текста ============================
20 f$ = "x Ungue LEONEM"
30 PRINT "Исходный текст"
40 PRINT "E" + f$
50 PRINT
```

```
'==2===== Блок вывода заглавных букв ==============================
60 f1$ = UCASE$(f$)
70 PRINT "Заглавные буквы"
80 PRINT "E" + f1$
90 PRINT
'==3===== Блок вывода строчных букв ==============================
100 f2$ = LCASE$(f$)
110 PRINT "Строчные буквы"
120 PRINT "E" + f2$
'=================================================================
130 END
```

Результат работы программы 10.16 показан на рис. 10.16.

Рис. 10.16. Программа, управляющая регистром



ГЛАВА 11

Подпрограммы

Использование в разрабатываемых программах логически завершенных блоков, оформленных в виде подпрограмм, отвечает принципам структурного программирования. Программа с применением подпрограмм становится значительно понятнее и компактнее, существенно ускоряется ее разработка.

Подпрограмма — это часть программы, которую можно выполнять многократно с различными значениями параметров.

Каждую подпрограмму можно разрабатывать и отлаживать отдельно от основной, иногда очень большой программы.

В BASIC различаются стандартные и пользовательские подпрограммы. Например, `COS(x)` является стандартной подпрограммой, специально созданной разработчиками среды программирования. Аргумент `x` называется *формальным* параметром. Он может приобрести конкретное, называемое *фактическим*, значение только при вызове функции в программе. Например, аргумент `x` примененных в следующем примере функций `TAN` и `COS` является уже фактическим параметром:

```
y = TAN(x) * COS(x)
```

Тип фактического параметра должен быть обязательно согласован с типом формального параметра. Имена (идентификаторы) функций могут быть операндами в математических выражениях, что видно также из приведенного примера.

Операндами называются объекты операций, реализуемых компьютером в ходе выполнения программы вычислений (это идентификаторы констант, переменных и функций).

В этой главе речь пойдет о разработке пользовательских подпрограмм.

По действиям интерпретатора подпрограммы `GOSUB...RETURN` отличаются от подпрограмм-функций (`FUNCTION`) и подпрограмм-процедур (`SUB`). Подпрограммы `GOSUB...RETURN` находятся в том же окне, что и основная программа. В случае же подпрограмм-функций и подпрограмм-процедур интерпретатор выделяет отдельные окна для основной программы и для каждой из подпрограмм. Переход из одного окна в другое окно осуществляется посредством клавиши <F2> (или с помощью пункта меню **Просмотр | SUBs** (View | SUBs)). При этом появляется диалоговое окно выбора, представленное на рис. 11.1.

Имя окна основной программы, совпадающее с именем программы, всегда находится в начале вертикального списка, а все подпрограммы упорядочены по алфавиту. Выбор требуемого окна осуществляется клавишами со стрелками вверх и вниз и клавишей <Enter>.

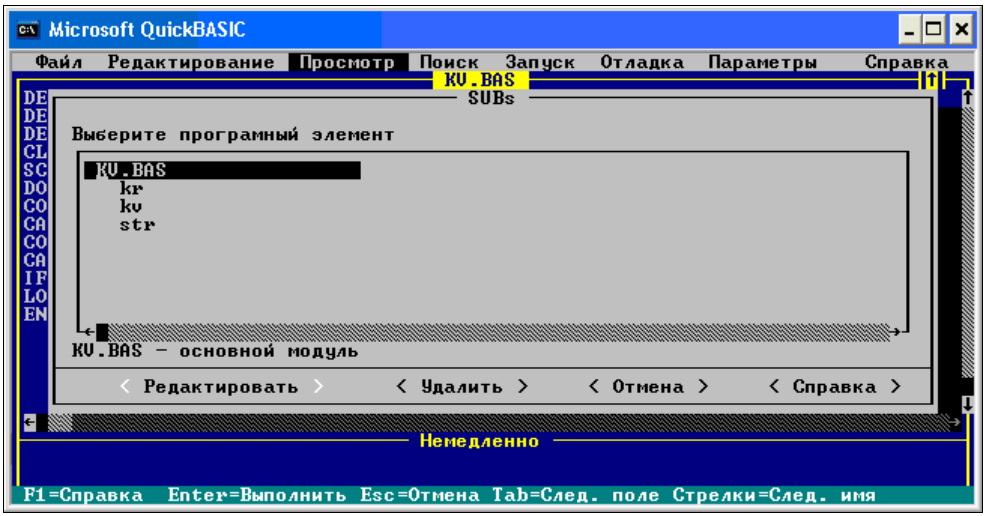


Рис. 11.1. Выбор окна с программой или подпрограммой

Прежде чем использовать в программе процедуру SUB или FUNCTION, необходимо уяснить отличие термина "параметр" от термина "аргумент".

Параметр — имя переменной, используемое в операторе SUB, FUNCTION или DECLARE.

Аргумент — это константа, переменная или выражение, которое передается в процедуру SUB или FUNCTION при их вызове.

Как показано на рис. 11.2, когда вызывается процедура, аргументы передаются в тело процедуры через заголовок, причем первый параметр получает значение первого аргумента, второй параметр получает значение второго аргумента и т. д.

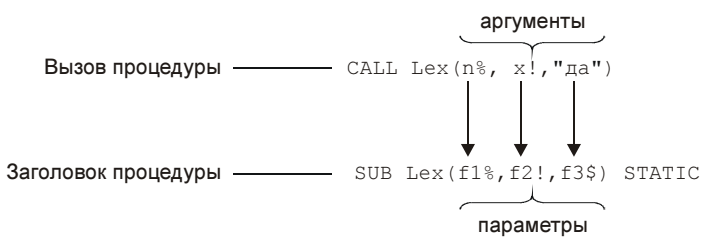


Рис. 11.2. Соответствие параметров и аргументов

На рис. 11.2 также демонстрируется еще одно важное правило. Хотя имена в списке аргументов и в списке параметров могут быть разными, но между аргументами и параметрами должно быть взаимно однозначное соответствие: их количество и типы должны совпадать.

Элементы, из которых могут состоять списки параметров и аргументов, приведены в табл. 11.1.

Подпрограмма FUNCTION, также как и SUB, имеет локальные переменные. Любая переменная подпрограммы, не входящая в список параметров, является локальной, за исключением переменных, описанных оператором SHARED внутри процедуры или операторами DIM или COMMON с атрибутом SHARED в главном модуле.

Таблица 11.1. *Соответствие элементов списка параметров списку аргументов*

Параметры	Аргументы
1. Имена переменных, кроме строк фиксированной длины	1. Константы
	2. Выражения
2. Имена массивов, которые записываются следующим образом: имя массива с пустыми скобками	3. Имена переменных
	4. Имена массивов (с пустыми скобками)

Чтобы сделать правильный выбор между подпрограммой-функцией и подпрограммой-процедурой, приведем основные различия между функциями и процедурами:

1. Функция, как правило, возвращает одно-единственное значение, например `COS(x)`. Процедура обычно производит какие-нибудь действия, например, сортирует массивы, выводит их на печать и пр.
2. Имя функции завершается суффиксом типа возвращаемого значения. В заголовке процедуры такая информация не требуется.
3. В конце тела функции ее имени присваивается значение, для вычисления которого она и предназначена, например, `Sum% = S`.
4. Функция может быть операндом математического выражения. Процедура не может быть частью выражения.

11.1. Подпрограммы-функции *FUNCTION*

Подпрограмма `FUNCTION` возвращает одиночное значение. Текст процедуры `FUNCTION` не входит в главную часть модуля.

`FUNCTION...END FUNCTION` — операторы, указывающие начало и конец подпрограммы `FUNCTION`.

```
FUNCTION imya (p1, ... , pn) STATIC
[операторы]
imya = f(p1, ... , pn)
[операторы]
END FUNCTION
```

где `imya` — имя подпрограммы, задаваемое как обычная переменная. Определяет тип возвращаемых данных;

`p1, ... , pn` — параметры — список переменных, значения которых передаются функции при ее вызове. Изменение значения параметра внутри функции изменит его значение в вызывающем модуле;

`f(p1, ... , pn)` — выражение, возвращаемое значение функции. Если имени функции ничего не присваивается, то возвращается значение по умолчанию: для числовых функций это ноль, для символьных — пустая строка;

`STATIC` — указывает, что локальные переменные функции сохраняются между вызовами. Иначе локальные переменные считаются динамическими и их значения теряются при выходе из функции. Атрибут `STATIC` не влияет на переменные, описанные вне функции операторами `DIM` или `COMMON` с атрибутом `SHARED`.

Подпрограмма FUNCTION принимает параметры, может состоять из серии операторов и изменять значения параметров. FUNCTION вызывается подобно встроенной функции языка BASIC.

Чтобы передать в процедуру в качестве аргумента массив, достаточно указать пустые скобки:

```
FUNCTION UmnMas (m$(), mas1(), mas2())
```

Для возвращения вычисленного значения из функции следует присвоить это значение имени функции.

Для вызова нового окна, предназначенного для разработки конкретной функции, необходимо выбрать пункт меню **Редактирование | Новая FUNCTION** (Edit | New FUNCTION).

В результате откроется диалоговое окно, представленное на рис. 11.3, в котором следует ввести имя функции, включающее суффикс типа возвращаемого значения.

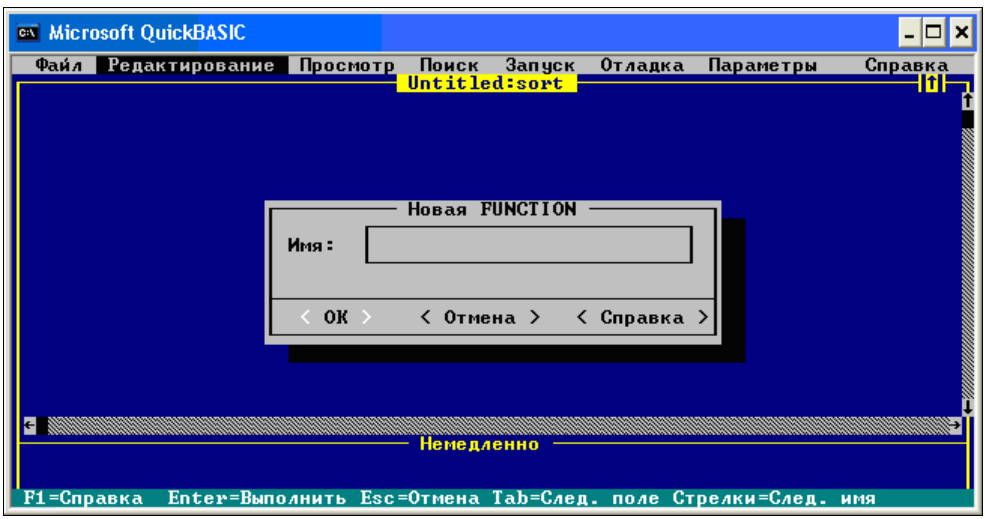


Рис. 11.3. Диалоговое окно имени новой FUNCTION

При этом имя подпрограммы должно быть правильным идентификатором, а параметры подпрограммы не должны вводиться. После нажатия кнопки **OK** или клавиши <Enter> появится заготовка для разработки функции с введенным ранее именем:

```
FUNCTION imya
END FUNCTION
```

Далее следует ввести сразу же после имени подпрограммы ее формальные параметры, если они у нее должны быть. Начиная со следующей строки, вводится тело самой подпрограммы. Не забывайте заканчивать его идентификатором, совпадающим с именем функции, и присвоить ему результат работы функции.

Вызвать новое окно с заготовкой функции также можно, записав заголовок функции в окне главного модуля и нажав клавишу <Enter>.

После сохранения созданной подпрограммы в окне главного модуля должен автоматически генерироваться оператор DECLARE. Обычно рекомендуется начинать программу с оператора очистки экрана CLS. Однако в программах с подпрограммами-функциями (как и с подпрограммами-процедурами) этого делать нельзя, поскольку оператор CLS является выполни-

мым, а перед оператором DECLARE могут быть только невыполнимые операторы. Все операторы BASIC являются выполнимыми, за исключением представленных в табл. 11.2.

Таблица 11.2. *Невыполнимые операторы BASIC*

№	Оператор	№	Оператор
1	COMMON	7	OPTION BASE
2	CONST	8	REM или апостроф
3	DATA	9	SHARED
4	DECLARE	10	STATIC
5	DEF *тип*	11	TYPE
6	DIM (для статических массивов)	12	$STATIC и $DYNAMIC

Рассмотрим несколько примеров.

Пример 11.1. Вычисление десятичного логарифма

Требуется разработать программу вычисления десятичного логарифма.

Программа 11.1

```
'111g10.bas
10 DECLARE  FUNCTION  lg (x)
20 CLS
30 INPUT "Введите число n = ", n
40 PRINT USING "Десятичный логарифм = #.##"; lg(n)
50 END
'==== Подпрограмма-функция вычисления десятичного логарифма ========
FUNCTION  lg (y)
lg = LOG(y) / LOG(10)
END FUNCTION
'=====================================================================
```

Результат работы программы 11.1 представлен на рис. 11.4.

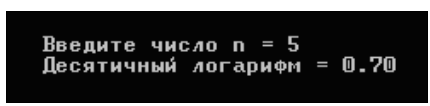


Рис. 11.4. Вычисление десятичного логарифма

Подпрограмма FUNCTION может использоваться в математических выражениях. Так, в формуле вычисления числа сочетаний факториал встречается три раза:

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

Покажем, как это отобразится в программе.

Пример 11.2. Вычисление числа сочетаний

Требуется разработать программу вычисления числа сочетаний.

Программа 11.2

```
'11coheta.bas
10 DECLARE  FUNCTION  faktrial (k)
20 CLS
30 INPUT "Введите число n = ", n                  'должно быть m < n
40 INPUT "Введите число m = ", m
50 C = faktrial(n) / (faktrial(m) * faktrial(n — m))
60 PRINT "Число сочетаний C = "; C
70 END
'===== Подпрограмма-функция вычисления факториала ==================
100 FUNCTION  faktrial (k)
110 P = 1
120 FOR i = 1 TO k
130 P = P * i
140 NEXT i
150 faktrial = P
160 END FUNCTION
'====================================================================
```

Результат работы программы 11.2 приведен на рис. 11.5.

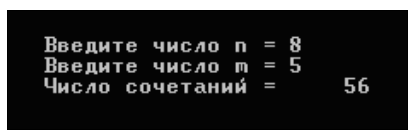


Рис. 11.5. Вычисление числа сочетаний

Результат вычисления в подпрограмме присваивается имени функции в строке 150. Подпрограмма вызывается в строке 50, причем имя функции выступает в качестве операндов формулы в строке 50.

Пример 11.3. Использование ключевого слова *STATIC*

Следующая программа иллюстрирует действие ключевого слова `STATIC` в подпрограмме-функции.

Программа 11.3

```
'11static.bas
10 DECLARE FUNCTION summa! (n%)
20 DECLARE FUNCTION sum! (n%)
30 CLS
40 LOCATE 2, 2: PRINT "STATIC"
50 FOR k = 1 TO 4
60 LOCATE k + 2, 3: PRINT sum(n%)
70 LOCATE k + 2, 10: PRINT summa(n%)
```

```
80 NEXT k
90 END
'==1===== Подпрограмма-функция со STATIC ==========================
FUNCTION sum (n%) STATIC
FOR i1 = 1 TO 5
s = s + 1
NEXT i1
sum = s
END FUNCTION
'==2===== Подпрограмма-функция без STATIC ==========================
FUNCTION summa (n%)
FOR i2 = 1 TO 5
s = s + 1
NEXT i2
summa = s
END FUNCTION
'==================================================================
```

Результат работы подпрограммы 11.3 приведен на рис. 11.6.

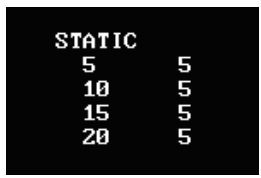


Рис. 11.6. Действие ключевого слова STATIC

Программа 11.3 производит суммирование 4 раза (цикл FOR...NEXT в строках 50—80) по 5 (в подпрограммах). Как нетрудно заметить, подпрограммы программы 11.3 отличаются друг от друга лишь именем и ключевым словом STATIC. Подпрограмма без ключевого слова STATIC каждый раз при обращении к ней обнуляет значение s, а подпрограмма со STATIC сохраняет предыдущее значение этой переменной.

Применение ключа STATIC слегка повышает скорость выполнения, но STATIC не используется с рекурсивными процедурами FUNCTION. Кроме того, необходимо самим заботиться о присвоении начальных значений всем переменным процедуры, т. к. их старые значения сохраняются между вызовами. Поэтому лучше без нужды не использовать ключ STATIC, возложив на BASIC отслеживание начальных значений переменных подпрограммы.

Процедура FUNCTION может быть рекурсивной, т. е. вызывать саму себя, и может использоваться вне модуля, где она определена. Это значит, что находящаяся во вспомогательном модуле процедура становится доступной любому модулю программы. Для этого следует добавить оператор DECLARE в те модули, где процедура FUNCTION используется.

Пример 11.4. Рекурсивная функция вычисления факториала

Требуется разработать программу вычисления факториала.

Программа 11.4

```
'11fact.bas
10 DECLARE FUNCTION fact! (k%)
```

```
20 CLS
30 INPUT "Задайте число от 0 до 10 — ", m%
40 IF 0 <= m% AND m% <= 10 THEN
     PRINT STR$(m%) + "! = "; fact(m%)
     ELSE
     PRINT "Вне диапазона": SLEEP 2: GOTO 20
     END IF
50 END
'===== Подпрограмма-функция вычисления факториала ==================
100 FUNCTION fact (k%) STATIC
110 IF k% > 0 THEN
      fact = k% * fact(k% — 1)
      ELSE
      fact = 1
130 END IF
140 END FUNCTION
'====================================================================
```

В отличие от подпрограммы-функции вычисления факториала в программе 11.2, подпрограмма-функция в программе 11.4 является рекурсивной, что следует из второй строки условного оператора `IF...THEN` (строка 110) в блочной форме.

Пример 11.5. Выбор слов заданной длины

Требуется разработать программу нахождения и вывода слов заданной длины.

Программа 11.5

```
'11vibor.bas
10 DECLARE FUNCTION dlina$ (m$(), dl%, i1k)
20 CLS
30 n = 25
40 DIM m$(n)
'==1===== Блок ввода слов ==========================================
50 FOR i = 1 TO n
60 DATA лев,лось,лиса,волк,овца,кот,вол,бык,бизон,кабан,кулан,лань
70 DATA тюлень,лошадь,косуля,пума,медведь,обезьяна,белка,горностай
75 DATA носорог,ондатра,выхухоль,аллегатор,слон
80 READ m$(i)
90 NEXT i
'==2===== Блок выбора слов по заданной длине =======================
100 i1k = n
110 INPUT "Введите длину слова (3-9) — ", dl%
120 DO
130 IF dlina$(m$(), dl%, i1k) = "да" THEN PRINT m$(i1k)
140 i1k = i1k — 1
150 IF dlina$(m$(), dl%, i1k) = "нет" THEN EXIT DO
160 LOOP
'====================================================================
170 END
```

```
'==3===== Подпрограмма определения длины слова ====================
200 FUNCTION dlina$ (m$(), dl%, i1k)
210 FOR i1 = 1 TO i1k
220 IF LEN(m$(i1)) = dl% THEN
      dlina$ = "да"
      SWAP m$(i1), m$(i1k)
      ELSE
      dlina$ = "нет"
230 END IF
240 NEXT i1
250 END FUNCTION
'==================================================================
```

Результат работы программы 11.5 дан на рис. 11.7.

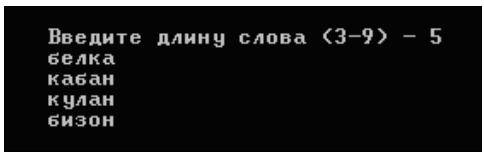


Рис. 11.7. Выбор слов заданной длины

Символьный массив `m$(n)`, вводимый посредством цикла `FOR...NEXT` в строках 50—90, передается в подпрограмму как имя массива с пустыми скобками (оператор `DECLARE` в строке 10). Вызов подпрограммы осуществляется в строках 130, 150 главного модуля программы, входящих в тело цикла `DO...LOOP` в строках 120—160. Программа 11.5 работает следующим образом. После ввода массива слов с помощью оператора `INPUT` в строке 110 задается длина слов для поиска. Затем подпрограмма, перебирая последовательно слова массива, определяет их длину. После нахождения слова заданной длины, оно меняется местами с последним элементом массива посредством оператора `SWAP` и выводится на экран оператором `PRINT` в строке 130. Затем в строке 140 уменьшается на единицу конечное значение счетчика цикла `FOR...NEXT` подпрограммы (чтобы исключить из рассмотрения поставленное в конец массива уже найденное слово заданной длины). И поиск в подпрограмме повторяется. Заканчивается работа, когда найдены все слова заданной длины. В этом случае имени функции присваивается значение `"нет"`, определяющее выход из цикла с помощью оператора `IF...THEN` в строке 150 основного модуля программы 11.5.

11.2. Подпрограммы-процедуры *SUB*

Для вызова нового окна, предназначенного для разработки конкретной процедуры, необходимо выбрать пункт меню **Редактирование | Новая SUB** (Edit | New SUB).

В результате откроется окно ввода, аналогичное представленному на рис. 11.3, в котором следует ввести имя процедуры.

При этом имя подпрограммы должно быть правильным идентификатором, а параметры подпрограммы не должны вводиться. После нажатия кнопки **ОК** или клавиши <Enter> появится заготовка для разработки процедуры с введенным ранее именем:

```
SUB imya
END SUB
```

Далее следует ввести сразу же после имени подпрограммы ее формальные параметры, если они у нее должны быть. Начиная со следующей строки, вводится тело самой подпрограммы.

Вызвать новое окно с заготовкой процедуры также можно, записав заголовок процедуры в окне главного модуля и нажав клавишу <Enter>.

После сохранения и выхода из созданной подпрограммы в окне главного модуля должен автоматически генерироваться оператор DECLARE.

SUB...END SUB — это оператор, указывающий начало и конец процедуры SUB:

```
SUB imya (p1, ... , pn) STATIC
[операторы]
[EXIT SUB]
[операторы]
END SUB
```

где imya — имя процедуры, переменная до 40 знаков. Это имя не должно использоваться другими операторами FUNCTION или SUB;

p1, ... , pn — список параметров, содержащий имена простых переменных и массивов, передаваемых в процедуру. Имена в списке разделяются запятыми. Любое изменение аргумента в процедуре приведет к изменению его значения в вызывающем модуле. Список параметров имеет следующую форму:

```
im1 AS tip, im2() AS tip, im3 AS tip
```

im1, im2, im3 — имя переменной или массива, причем массив задается с пустыми скобками; tip — тип переменной. Может быть INTEGER, LONG, SINGLE, DOUBLE, STRING или пользовательским. Не может быть фиксированной строкой. Строки фиксированной длины необходимо преобразовать в строки переменной длины.

Необязательный атрибут STATIC указывает, что все локальные для процедуры SUB переменные являются статическими, т. е. их значения сохраняются между вызовами. Использование ключа STATIC несколько повышает скорость выполнения процедуры, в то же время STATIC не используется в рекурсивных процедурах.

SUB и END SUB отмечают начало и конец процедуры. Для выхода из процедуры можно использовать оператор EXIT SUB. Внутри процедуры SUB нельзя определять процедуры SUB и FUNCTION, функции DEF FN, а также использовать операторы GOTO, GOSUB, RETURN для входа в процедуру или выхода из процедуры.

Все переменные подпрограммы являются локальными, кроме описанных операторами DIM, REDIM, COMMON как SHARED в главном модуле или оператором SHARED в самой процедуре.

Процедуры вызываются оператором CALL или путем указания их имени со списком аргументов. SUB используется подобно операторам языка BASIC. Процедура SUB, также как и FUNCTION, может быть рекурсивной.

Рассмотрим несколько примеров, поясняющих применение подпрограмм-процедур в программах.

Пример 11.6. Динамическая смена дня и ночи

Требуется разработать программу динамической картинки смены дня и ночи.

Программа 11.6

```
'11DenNoh.bas
10 DECLARE SUB luna (a!)
20 DECLARE SUB sol (a!)
30 CLS
40 SCREEN 9
50 DO
60 COLOR 14, 11
70 CALL sol(a!)                 'вызов подпрограммы движения солнца
80 COLOR 14, 1
90 CALL luna(a!)                'вызов подпрограммы движения луны
100 IF INKEY$ = CHR$(27) THEN EXIT DO             'клавиша Esc
110 LOOP
120 END
'==1===== Подпрограмма движения луны ==============================
200 SUB luna (a!)
210 FOR a = 3.2 TO 0 STEP -.01
220 x = 320 + 200 * COS(a)
230 y = 250 — 200 * SIN(a)
240 CIRCLE (x, y), 20, 14, -1.57, -4.71           'рисует луну
250 DRAW "bl18 p14,14"
260 FOR iv = 1 TO 5000: NEXT iv                   'выдержка времени
270 DRAW "p0,0"                                   'стирает луну
280 NEXT a
290 END SUB
'==2===== Подпрограмма движения солнца ============================
300 SUB sol (a!)
310 FOR a = 3.2 TO 0 STEP -.01
320 CIRCLE (320 + 200 * COS(a), 250 — 200 * SIN(a)), 20, 14
                                                  'рисует солнце
330 DRAW "p14,14"
340 FOR iv = 1 TO 5000: NEXT iv                   'выдержка времени
350 DRAW "p0,0"                                   'стирает солнце
360 NEXT a
370 END SUB
'==================================================================
```

Сюжет программы 11.6 следующий: на фоне светлого неба по дуге прокатывается солнце, затем по темному небу по той же дуге движется луна. И все повторяется.

В главном модуле программы создан цикл `DO...LOOP`, в теле которого в строках 70, 90 оператором `CALL` вызываются подпрограммы движения солнца и луны. В качестве параметра в подпрограммы передается аргумент `a` — угол, который в подпрограммах меняется приблизительно от `PI` до 0.

Результат работы программы 11.6 показан на рис. 11.8.

Массивы в качестве данных передаются в подпрограмму-процедуру так же, как и в процедуру-функцию — имя массива с пустыми скобками.

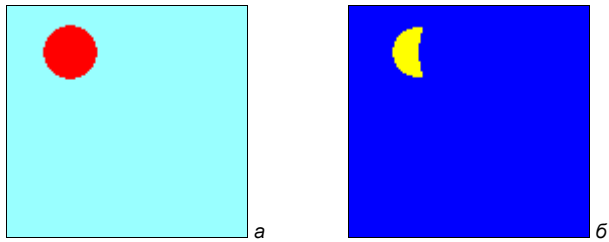


Рис. 11.8. Смена дня и ночи: Солнце на дневном небе (*а*); Луна на ночном небе (*б*)

Пример 11.7. Сортировка строковых массивов

Требуется разработать программу сортировки по длине слов и вывода на экран отсортированных символьных массивов.

Программа 11.7

```
'11strmas.bas
10 DECLARE SUB sort (m1$(), m2$(), m3$(), i)
20 DECLARE SUB vivod (m1$(), m2$(), m3$(), i)
30 CLS
40 DIM m1$(5), m2$(5), m3$(5)
'==1===== Блок ввода массивов ======================================
50 FOR i = 1 TO 3
60 PRINT "Введите"; i; "-й массив"
70 FOR j = 1 TO 5
80 PRINT "m"; i; "("; j; ")";
90 IF i = 1 THEN INPUT " = ", m1$(j)
100 IF i = 2 THEN INPUT " = ", m2$(j)
110 IF i = 3 THEN INPUT " = ", m3$(j)
120 NEXT j
'---------- Вызов подпрограмм -------------------------------------
130 CALL sort(m1$(), m2$(), m3$(), i)
140 CALL vivod(m1$(), m2$(), m3$(), i)
'------------------------------------------------------------------
150 NEXT i
160 END
'==2===== Подпрограмма-процедура сортировки =======================
200 SUB sort (m1$(), m2$(), m3$(), i)
210 FOR js = 1 TO 4
220 FOR ks = js + 1 TO 5
230 SELECT CASE i
    CASE 1
IF LEN(m1$(js)) > LEN(m1$(ks)) THEN SWAP m1$(ks), m1$(js)
    CASE 2
IF LEN(m2$(js)) > LEN(m2$(ks)) THEN SWAP m2$(ks), m2$(js)
    CASE 3
IF LEN(m3$(js)) > LEN(m3$(ks)) THEN SWAP m3$(ks), m3$(js)
240 END SELECT
250 NEXT ks, js
```

```
260 PRINT i
270 END SUB
'==3===== Подпрограмма-процедура вывода результата ================
300 SUB vivod (m1$(), m2$(), m3$(), i)
310 CLS
320 FOR iv1 = 1 TO
330 PRINT iv1; "-й массив"
340 FOR iv = 1 TO 5
350 IF iv1 = 1 THEN PRINT m1$(iv); " ";
360 IF iv1 = 2 THEN PRINT m2$(iv); " ";
370 IF iv1 = 3 THEN PRINT m3$(iv); " ";
380 NEXT iv
390 PRINT
400 NEXT iv1
410 END SUB
'================================================================
```

Программа 11.7 работает следующим образом. После ввода с клавиатуры очередного массива слов управление передается в подпрограмму-процедуру сортировки, где массив упорядочивается по длине слов, а затем с помощью подпрограммы-процедуры вывода передается на экран. Всего вводится 3 массива. В главном модуле программы организован двойной (вложенный) цикл `FOR...NEXT` в строках 50—150. Вложенный цикл в строках 70—120 непосредственно служит для ввода слов массивов с клавиатуры, а внешний цикл со счетчиком `i` обеспечивает переход от массива к массиву и вызов подпрограмм (строки 130—140). Символьные массивы передаются в подпрограммы путем указания их имен с пустыми скобками.

Чтобы лучше понять разницу между глобальными и локальными переменными, попробуйте поэкспериментировать с программой 11.7. Уберите переменную `i` в строках 10—20 и 130—140, приведя их к следующему виду:

```
 10 DECLARE SUB sort (m1$(), m2$(), m3$())
 20 DECLARE SUB vivod (m1$(), m2$(), m3$())
  ...
130 CALL sort(m1$(), m2$(), m3$())
140 CALL vivod(m1$(), m2$(), m3$())
  ...
```

Не забудьте также убрать `i` из заголовков подпрограмм. Проанализировав полученный результат, измените строку 40 следующим образом:

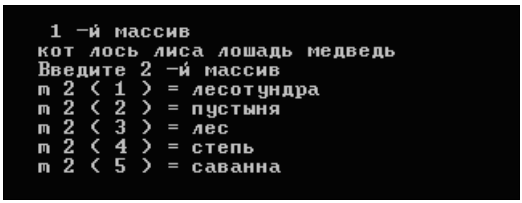
```
 40 DIM SHARED i AS INTEGER, m1$(5), m2$(5), m3$(5)
```

Результат работы программы 11.7 показан на рис. 11.9.

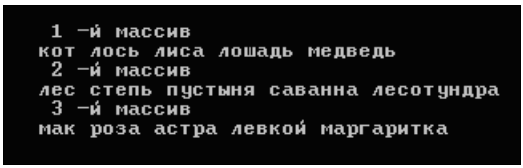
Подпрограммы-процедуры могут быть без параметров. Переменные главного модуля в этом случае следует делать видимыми в подпрограмме, объявляя их оператором `DIM` с ключевым словом `SHARED`. Покажем это на примере.

Была такая игра в "ромбы". Пусть участвуют 14 человек. На листочках бумаги писались числа, например от 100 до 700 с шагом 100 (на 7 человек), каждая в двух экземплярах. Одни "ромбы" помечались, например, крестами — одна команда, другие звездочками — другая команда. Если встречались два противника из разных команд, то они предъявляли друг другу свои ромбы (в процессе игры их могло уже быть несколько у одного на руках). У кого

число было больше, тот забирал "ромб" противника, увеличивая таким образом свой "вес". Побеждала команда, собравшая все "ромбы" в одних руках. Если игроки из разных команд набирали равное количество очков, то фиксировалась ничья.



а



б

Рис. 11.9. Подпрограмма ввода и упорядочивания 3-х строковых массивов: этап ввода второго массива (*а*); этап окончания работы программы (*б*)

Пример 11.8. Игра в "ромбы"

Разработать программу, имитирующую игру в "ромбы".

Программа 11.8

```
'11rombi.bas
10 DECLARE SUB hod ()
20 CLS
30 DIM SHARED x(7), y(7)
40 RANDOMIZE 32767 - TIMER
'==1===== Блок ввода ============================================
50 FOR i = 1 TO 7
60 x(i) = i * 100
70 y(i) = i * 100
80 NEXT i
'==2===== Блок перемешивания =====================================
90 n = 7
100 FOR i = 1 TO 6
110 k1 = INT(1 + n * RND(1))
120 k2 = INT(1 + n * RND(1))
130 SWAP x(k1), x(n)
140 SWAP y(k2), y(n)
150 n = n - 1
160 NEXT i
'==3===== Блок игры ==============================================
170 DO
180 CALL hod
'--- Условие выхода из цикла "красные" ----------------------------
190 s1 = 0
200 FOR j1 = 1 TO 7
210 s1 = s1 + x(j1)
220 NEXT j1
230 IF s1 = 0 THEN EXIT DO
'--- Условие выхода из цикла "синие" -----------------------------
240 s2 = 0
250 FOR j2 = 1 TO 7
```

```
260 s2 = s2 + y(j2)
270 NEXT j2
280 IF s2 = 0 THEN EXIT DO
'-----------------------------------------------------------------
290 SLEEP
300 LOOP
'=================================================================
900 END

'==========  Подпрограмма ========================================
SUB hod STATIC
500 IF e = 0 THEN n1 = 7: n2 = 7: e = 1: GOTO 630
'==1===== Блок очистки сообщений ===================================
510 FOR k = 1 TO 6
520 LOCATE k, 1
530 PRINT "                                                        "
540 NEXT k
'==2===== Блок выбора пары и анализа выбора ========================
550 k3 = INT(1 + n1 * RND(1))
560 k4 = INT(1 + n2 * RND(1))
570 LOCATE 5, 4: PRINT k3; x(k3)
580 LOCATE 6, 4: PRINT k4; y(k4)
590 SLEEP 2
'---- Анализ значений ромбов ---------------------------------------
600 IF x(k3) > y(k4) THEN
     x(k3) = x(k3) + y(k4)
     y(k4) = 0
     SWAP y(k4), y(n2)
     n2 = n2 - 1
     END IF
610 IF x(k3) = y(k4) THEN 630
620 IF y(k4) > x(k3) THEN
   y(k4) = y(k4) + x(k3)
     x(k3) = 0
     SWAP x(k3), x(n1)
     n1 = n1 - 1
END IF
'==3===== Блок распечатки ==========================================
630 LOCATE 1, 1: PRINT "Красные"
640 LOCATE 2, 1
650 s1 = 0
660 FOR i1 = 1 TO n1
670 PRINT x(i1);
680 IF i1 = n1 THEN PRINT
690 NEXT i1
'-----------------------------------------------------------------
700 LOCATE 3, 1: PRINT "Синие"
710 LOCATE 4, 1
720 s2 = 0
730 FOR i2 = 1 TO n2
740 PRINT y(i2);
```

```
750 IF i2 = n2 THEN PRINT
760 NEXT i2
'=============================================================
END SUB
```

Результат работы программы 11.8 приведен на рис. 11.10.

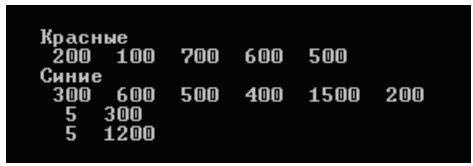


Рис. 11.10. Программа, имитирующая игру в "ромбы"

На рис. 11.10 можно видеть экран на промежуточном этапе после встречи пятого игрока команды "Красных", имевшего "ромб" достоинством 300, с пятым игроком "Синих", имевшего "ромбы" суммарным достоинством 1200. После этого у первого стало 0 очков, а у второго — 1500. У "Красных" осталось 5 активных игроков, а у "Синих" — 6.

В программе 8 использована подпрограмма-процедура без параметров, что видно из строк 10, 180 главного модуля и заголовка подпрограммы. В операторе DIM в строке 30, описывающим числовые массивы x(7), y(7), поставлено ключевое слово SHARED, делающее данные массивов видимыми в подпрограмме. В блоке ввода главного модуля в строках 50—80 осуществляется упорядоченный ввод "ромбов" всех достоинств для обеих команд. В блоке перемешивания главного модуля в строках 90—160 происходит перемещение "ромбов" в соответствии с данными, получаемыми с помощью генераторов случайных чисел — строки 110—120. В блоке игры главного модуля в строках 170—300 организован цикл DO...LOOP, в теле которого осуществляется вызов подпрограммы в строке 180, а также проверка на окончание игры. Игра заканчивается, если у одной из команд уже нет "ромбов", т. е. сумма их значений равна 0. Оператор SLEEP в строке 290 производит техническую задержку, продолжение — нажатием любой клавиши.

В подпрограмме осуществляется удаление предыдущих сообщений оператором PRINT в строке 530, выводящим строку пробелов. Затем случайным образом определяется пара контактирующих противников. Блок анализа определяет результаты контакта. Тот, у которого "ромб" больше, приплюсовывает себе очки противника, оставляя его с 0 (операторы IF...THEN в блочной форме в строках 600, 620). Если же значения "ромбов" равны, то все остается без изменения (оператор IF...THEN в строке 610). Далее результаты выводятся на экран.

11.3. Подпрограммы *GOSUB...RETURN*

GOSUB...RETURN — управляющие операторы вызова подпрограммы и выхода из него:

```
GOSUB m1
[операторы]
RETURN m2
```

где m1 — метка первой строки подпрограммы;

m2 — метка строки, куда возвращается управление после выхода из подпрограммы.

Подпрограмму GOSUB...RETURN можно вызывать любое количество раз. Можно вызвать одну подпрограмму из другой. В то же время подпрограмма GOSUB...RETURN не может быть рекурсивной, т. е. вызывать сама себя. Одна подпрограмма может иметь несколько операторов RETURN.

GOSUB и RETURN должны находиться в одном модуле. RETURN не может быть использован для передачи управления из одной подпрограммы типа SUB или FUNCTION другой подпрограмме.

Подпрограмма GOSUB...RETURN может располагаться в любом месте главного модуля. Обычно ее располагают в конце программы перед оператором END. Чтобы избежать обращение к подпрограмме без оператора GOSUB, перед подпрограммой ставится оператор GOTO. Поясним это на примере.

Пример 11.9. Ньютон и яблоко

Требуется разработать программу, реализующую динамическую картинку на сюжет "Ньютон и яблоко".

Программа 11.9

```
'11Nyuton.bas    Ньютон
5 CLS
10 SCREEN 9
15 COLOR 14
'==1=== Блок рисования яблони ======================================
20 LINE (210, 10)-(230, 320), 6, BF           'Ствол
25 FOR y1 = 20 TO 120 STEP 25                 'Цикл рисования ветвей
30 FOR x1 = 100 TO 240 STEP 140
35 IF x1 = 100 AND (y1 = 20 OR y1 = 70 OR y1 = 120) THEN GOSUB 200
40 IF x1 = 240 AND (y1 = 45 OR y1 = 95) THEN GOSUB 200
45 NEXT x1, y1
'==2=== Блок рисования Ньютона =====================================
50 CIRCLE (288, 321), 20, 14, -.01, -1.57, .3               'Ступня
55 PAINT (290, 318), 14, 14
60 DRAW "bm274,284 c14 m+2,-2 r10 m+2,+2 m+10,+37"          'Ноги
65 DRAW "l10 m-10,-20 m-15,+15 u23 m+11,-9 bm278,284 p14,14"
70 CIRCLE (270, 242), 4, 14: PAINT (270, 242), 14, 14       'Нос
75 CIRCLE (250, 242), 19, 14: PAINT (250, 242), 14, 14      'Голова
80 CIRCLE (260, 237), 3, 1: PAINT (260, 237), 1, 1          'Глаз
85 LINE (267, 247)-(260, 249), 12, BF                       'Губы
90 CIRCLE (250, 268), 19, 14: PAINT (250, 268), 14, 14      'Туловище
95 LINE (231, 265)-(269, 295), 14, BF
100 CIRCLE (250, 307), 18, 1: PAINT (250, 307), 1, 1        'Трусы
105 DRAW "bm268,312 c1 m+8,-8 m-7,-17 m-36,+10 bm+34,-8 p1,1 "
110 LINE (231, 311)-(269, 294), 1, BF
115 DRAW "bm294,296 c7 m251,288 m-3,-3 m245,269 m+2,-2"     'Рука
120 DRAW "r6 m+2,+2 m+5,15 m+32,+6 m294,296 bm-3,-3 p14,7"
'==3=== Блок падения яблока ========================================
125 WHILE INKEY$ <> CHR$(27)                        'Клавиша Esc
130 x = 260: y = 119: c = 0: a = 0: SLEEP 2
135 WHILE c <> 1
140 CIRCLE (x, y), 10, 12: PAINT (x, y), 12, 12    'Рисует яблоко
```

```
145 FOR i = 1 TO 1500: NEXT i                    'Временная задержка
150 CIRCLE (x, y), 10, 0: PAINT (x, y), 0, 0     'Стирает яблоко
155 IF a = 0 THEN y = y + 1 ELSE y = y + 1: x = x + 3
160 IF b = 0 AND y = 221 THEN a = 1
165 IF b = 0 AND y = 320 THEN c = 1
170 IF b = 1 AND y = 221 THEN GOSUB 300
175 IF b = 1 AND y = 230 THEN c = 1
180 WEND
185 IF b = 1 THEN GOSUB 400
190 IF b = 0 THEN CIRCLE (x, y), 10, 12: PAINT (x, y), 12, 12
195 LOCATE 5, 54: PRINT "Э В Р И К А  !!!": b = 1
196 WEND
197 GOTO 500
'==4=== Подпрограмма рисования ветки ==============================
200 x2 = x1 + 4 * 20
205 LINE (x1, y1)-(x1 + 100, y1 + 5), 6, BF              'Ветка
210 FOR y = y1 TO y1 + 14 STEP 14            'Цикл рисования листьев
215 FOR x = x1 TO x2 STEP 20
220 CIRCLE (x + 10, y - 4), 5, 10, , , 1.1
225 PAINT (x + 10, y - 4), 10, 10
230 NEXT x, y
235 FOR x = x1 TO x1 + 60 STEP 60            'Цикл рисования яблок
240 CIRCLE (x + 20, y - 4), 10, 12
245 PAINT (x + 20, y - 4), 12, 12
250 NEXT x
255 RETURN
'==5=== Голова с открытым ртом ====================================
300 LINE (230, 223)-(274, 256), 0, BF
310 CIRCLE (246, 228), 4, 14: PAINT (246, 228), 14, 14      'Нос
320 CIRCLE (250, 242), 19, 14: PAINT (250, 242), 14, 14     'Голова
330 LINE (250, 242)-(270, 222), 0, BF
340 CIRCLE (240, 237), 3, 1: PAINT (240, 237), 1, 1         'Глаз
350 RETURN
'==6=== Голова с закрытым ртом ====================================
400 LINE (230, 223)-(274, 256), 0, BF
410 CIRCLE (270, 242), 4, 14: PAINT (270, 242), 14, 14      'Нос
420 CIRCLE (250, 242), 19, 14: PAINT (250, 242), 14, 14     'Голова
430 CIRCLE (260, 237), 3, 1: PAINT (260, 237), 1, 1         'Глаз
440 LINE (267, 247)-(260, 249), 12, BF                      'Губы
450 RETURN
'====================================================================
500 END
```

Результат работы программы 11.9 дан на рис. 11.11.

Реализованный сюжет заключается в следующем. Первое яблоко бьет Ньютона по голове и откатывается в сторону, в результате чего тот понимает, что яблоки лучше есть (рис. 11.11, *а*). Другие падающие яблоки Ньютон уже глотает (рис. 11.11, *б*).

Программа содержит 3 подпрограммы, находящиеся перед оператором END. Поэтому в строке 197 стоит оператор GOTO 500, позволяющий перепрыгнуть на END. На яблоне 5 веток, поэтому блок рисования ветки выгоднее выделить в подпрограмму. Когда яблоко подлетает к голове, то голову с закрытым ртом сменяет голова с разинутым ртом. Яблоко стирается,

создавая иллюзию того, что его проглотили. После чего голова с закрытым ртом возвращается на плечи Ньютона и т. д. Поэтому для реализации динамической картинки требуются подпрограммы, рисующие голову с открытым и закрытым ртом.

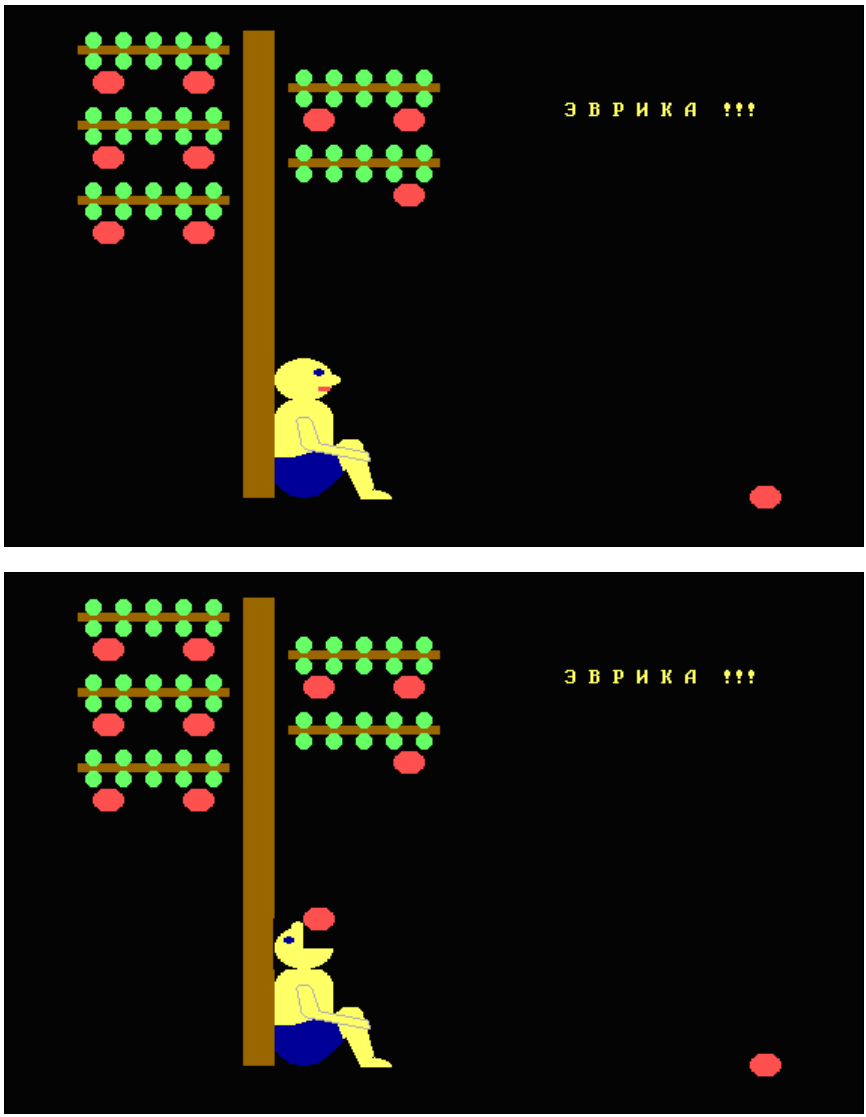


Рис. 11.11. Ньютон под яблоней

Подпрограмму допускается располагать и после оператора END, что иллюстрирует следующий пример.

Пример 11.10. Стража, шагающая по стене крепости

Требуется разработать программу, реализующую динамическую картинку стражи, шагающей по стене крепости.

Программа 11.10

```
'11ctraja.bas    Стража
10 CLS
20 SCREEN 9
30 COLOR 14, 9
'==1===== Блок рисования крепости ================================
40 LINE (0, 300)-STEP(650, 370), 2, BF
50 DRAW "BM0,140 C8 ND160 R40 u80 l10 u40 r20 d10 r10 u10 r10 d10 "
60 DRAW "r10 u10 r10 d10 r10 u10 r10 d10 r10 u10 r20 d40 l10 d80 R380"
70 DRAW "u80 l10 u40 r20 d10 r10 u10 r10 d10 r10 u10 r10 d10 r10 u10 r10"
80 DRAW "d10 r10 u10 r20 d40 l10 d80 R40 D160 L640 BE10 P7,8"
     '----- Ворота -----------------------------------------------
90 CIRCLE (320, 300), 140, 8, -.001, -3.14, 1.5
100 PAINT STEP(0, -2), 8, 8
110 CIRCLE STEP(0, 2), 110, 6, -.001, -3.14, 1.5
120 PAINT STEP(0, -2), 6, 6
130 LINE (319, 300)-STEP(2, -120), 8, BF
'==2===== Блок движения стражников -==============================
140 e1 = 1
150 x1 = 280: dx1 = -1
160 x2 = 360: dx2 = 1
170 DO
180 xc = x1: e = e1: GOSUB 500
190 xc = x2: e = -e1: GOSUB 500
200 IF x1 = 170 OR x1 = 290 THEN e1 = -e1: dx1 = -dx1: dx2 = -dx2
210 IF INKEY$ = CHR$(27) THEN EXIT DO                    'клавиша Esc
220 FOR iv = 1 TO 10000: NEXT iv                      'выдержка времени
230 LINE (131, 139)-STEP(378, -43), 0, BF
240 x1 = x1 + dx1
250 x2 = x2 + dx2
260 LOOP
270 END

'==3===== Подпрограмма рисования стражников =======================
500 cv1 = 8
510 CIRCLE (xc, 120), 5, cv1
520 PAINT STEP(0, 0), cv1, cv1
530 CIRCLE STEP(0, 8), 5, cv1
540 PAINT STEP(0, 0), cv1, cv1
550 LINE STEP(-5, 0)-STEP(10, 11), cv1, BF
560 IF e = -1 THEN
       LINE STEP(0, -5)-STEP(7, 3), cv1, BF
       DRAW "nh40 l1 nh40"
       ELSE
       LINE STEP(-10, -5)-STEP(-7, 3), cv1, BF
       DRAW "ne40 l1 ne40"
570 END IF
580 RETURN
'================================================================
```

По сюжету стражники, шагающие по стенам крепости и держащие копья на плечах, то сходятся, то расходятся. Подпрограмма, рисующая стражников, находится после оператора `END`. Условный оператор `IF...THEN` в блочной форме в строке 560 подпрограммы определяет с помощью переключателя `e` различия в рисовании фигуры правого и левого стражника. В блоке движения стражников организован цикл `DO...LOOP` в строках 170—260, в котором происходит обращение к подпрограмме посредством операторов `GOSUB` в строках 180 и 190.

Результат работы программы 11.10 приведен на рис. 11.12.

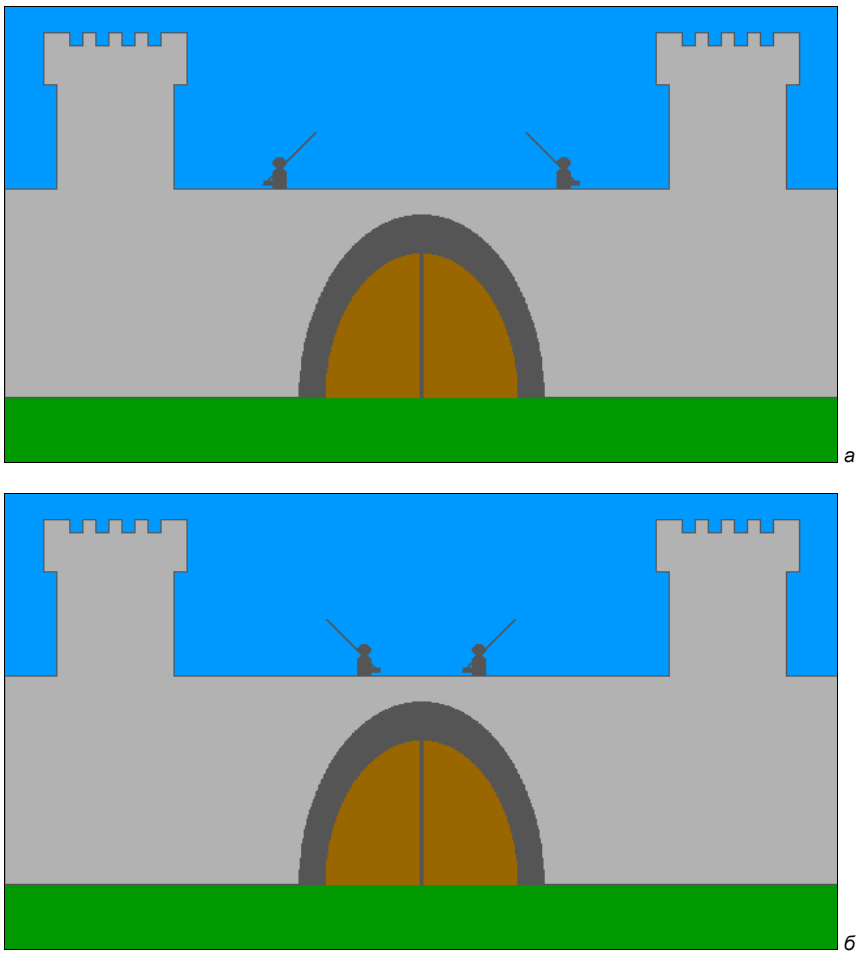


Рис. 11.12. Расходящаяся (*а*) и сходящаяся (*б*) стража

Если подпрограмм в программе достаточно много, то целесообразно использовать оператор `ON...GOSUB`.

`ON...GOSUB` — управляющий оператор вызова подпрограммы в зависимости от числового выражения:

```
ON f% GOSUB 1, 2, ... , n
```

где `f%` — выражение в диапазоне от 0 до 255;

1, 2, ... , n — набор меток или номеров строк. Если значение выражения равно 1, то программа переходит на первую метку или в первую строку из списка; если выражение равно 2, то программа переходит на вторую метку или во вторую строку из списка и т. д.

Поясним применение оператора ON...GOSUB на примере.

Пример 11.11. Использование оператора *ON...GOSUB*

Требуется разработать программу выбора остроумного изречения на заданную тему.

Программа 11.11

```
'11frazi.bas
10 CLS
20 PRINT : PRINT " Изречения французов"
30 PRINT " 1 — О дружбе    2 — О деньгах   3 — О демократии"
40 PRINT " 4 — О дураках   5 — О доверии   6 — О микробах "
50 PRINT
60 INPUT " Ваш выбор — ", fr
70 ON fr GOSUB 1, 2, 3, 4, 5, 6
80 PRINT
90 INPUT " Желаете еще (Да — Enter, нет — 1) ", p
100 IF p = 1 THEN 110 ELSE 60
110 END
'==1===== О дружбе ================================================
1 f1$ = " Я узнаю о том, что мой друг преуспел, "
  f2$ = "по тому, что он теряет память."
  PRINT f1$ + f2$
  PRINT " Оноре де Бальзак"
  RETURN
'==2===== О деньгах ===============================================
2 f1$ = " Деньги дают все то, "
   f2$ = "что другим представляется счастьем. "
   PRINT f1$ + f2$
   PRINT " Анри де Ренье"
   RETURN
'==3===== О демократии ============================================
3 f1$ = " Демократия? Знаете, что это такое? "
   f2$ = "Это право вшей поедать львов."
   PRINT f1$ + f2$
   PRINT " Жорж Клемансо"
   RETURN
'==4===== О дураках ===============================================
4 f1$ = " Наличие разума у дурака, "
   f2$ = "- вот что пугает больше всего."
   PRINT f1$ + f2$
   PRINT " Анатоль Франс"
   RETURN
'==5===== О доверии ===============================================
5 f1$ = " Часто доверие — разновидность лени, потому что "
   f2$ = "доверять легче, чем проверять."
   PRINT f1$ + f2$
```

```
   PRINT " Фернан Вадран"
   RETURN
'==6===== О микробах ============================================
6 f1$ = " У микроба нет времени изучать биолога."
   PRINT f1$
   PRINT " Анри Мишо"
   RETURN
'================================================================
```

Результат работы программы 11.11 дан на рис. 11.13.

```
Изречения французов
1 - О дружбе    2 - О деньгах   3 - О демократии
4 - О дураках   5 - О доверии   6 - О микробах

Ваш выбор - 2
Деньги дают все то, что другим представляется счастьем.
Анри де Ренье

Желаете еще (Да - Enter, нет - 1)
Ваш выбор - 3
Демократия? Знаете, что это такое? Это право вшей поедать львов.
Жорж Клемансо

Желаете еще (Да - Enter, нет - 1)
Ваш выбор - 6
У микроба нет времени изучать биолога.
Анри Мишо

Желаете еще (Да - Enter, нет - 1) 1
```

Рис. 11.13. Программа с оператором ON...GOSUB

В строках 20—50 программы 11.11 организуется своеобразное меню-подсказка, определяющее выбор оператором INPUT в строке 60. Выбор обрабатывается оператором ON...GOSUB в строке 70, передающим управление в соответствующую подпрограмму.

11.4. Функция *DEF FN*

DEF FN — оператор, описывающий функцию как пользовательскую. Существуют линейная и блочная формы этого оператора.

Линейная форма:

```
DEF FNimya(p1, ... ,pn) = f(p1, ... ,pn)
```

Блочная форма:

```
DEF FNimya(p1, ... ,pn)
[операторы]
FNimya = f(p1, ... ,pn)
[операторы]
[EXIT DEF]
[операторы]
END DEF
```

где imya — имя переменной до 40 знаков. Комбинируемое с "FN" имя является именем функции. Имя может иметь суффикс определения типа, указывающий на тип возвращаемо-

го значения. Для присвоения значения функции нужно присвоить его полному имени так, как обычной переменной:

```
FNString$ = "да"
```

p1, ... , pn — параметры, список переменных, разделенных запятыми. При вызове функции значения каждого аргумента присваиваются соответствующему параметру. Аргументы передаются по значению. DEF FN не поддерживает массивы, записи или символьные строки фиксированной длины в качестве аргументов.

f(p1, ... , pn) — выражение, вычисляющее результат функции. В линейном синтаксисе выражение является телом функции. Если нет выражений, присвоенных имени функции, то по умолчанию для числовой функции возвращается 0, а для символьной функции — пустая строка.

Перед использованием функция должна быть определена, иначе при ее вызове появится сообщение об ошибке "Функция не определена" (FUNCTION not defined).

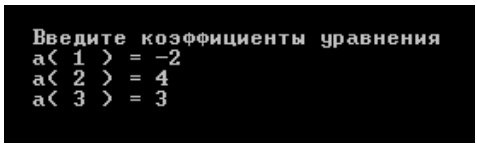
Для выхода до ее завершения из функции в блочной форме следует использовать оператор EXIT DEF. Функция может вызываться только из того модуля, в котором она определена. Обращения из других модулей недействительны. Поэтому если в программе несколько модулей, то следует поместить определение функции DEF FN в каждый модуль. Функция DEF FN не проверяет тип передаваемого ей числового аргумента, что позволяет использовать функции DEF FN, которым можно передавать любой числовой аргумент.

Пример 11.12. Решение квадратного уравнения

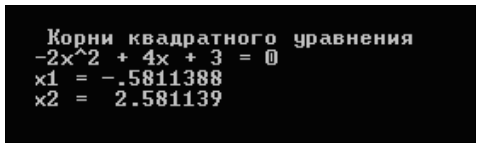
Требуется разработать программу решения квадратного уравнения.

Программа 11.12

```
 '11DEFfn.bas
10 CLS
20 DIM a$(3), b(3)
30 z$(2) = " +"
40 z$(3) = " +"
'==1===== Блок ввода коэффициенты уравнения ========================
50 PRINT "Введите коэффициенты уравнения"
60 FOR i = 1 TO 3
70 PRINT "a("; i; ")";
80 INPUT " = ", a(i)
90 b(i) = a(i)
100 IF i > 1 AND a(i) < 0 THEN a(i) = ABS(a(i)): z$(i) = " -"
110 a$(i) = STR$(a(i))
120 NEXT i
'==2===== Блок вывода результата ==================================
130 CLS
140 PRINT " Корни квадратного уравнения"
150 PRINT a$(1) + "x^2" + z$(2) + a$(2) + "x" + z$(3) + a$(3) + " = 0"
160 DEF fnx (b) = SQR(b(2) ^ 2 - 4 * b(1) * b(3)) / (2 * b(1))
170 c = -b(2) / (2 * b(1))
180 PRINT "x1 = "; c + fnx(b)
190 PRINT "x2 = "; c - fnx(b)
'==================================================================
200 END
```



а



б

Рис. 11.14. Программа решения квадратного уравнения: этап ввода коэффициентов (*а*); вывод уравнения и найденных корней (*б*)

Результат работы программы 11.12 показан на рис. 11.14.

Чтобы использовать цикл (цикл `FOR...NEXT` в строках 60—120), коэффициенты квадратного уравнения вводятся как `a(i)` оператором `INPUT` в строке 80, которые потом дублируются как `b(i)` в строке 90. Чтобы вывести на экран само квадратное уравнение, `a(i)` переводятся в строковые оператором `STR$` в строке 110, тогда как `b(i)` используются в формуле при вычислениях. Первая часть формулы для корней уравнения определяется как `c` в строке 170 посредством оператора присваивания. Вторая же часть определяется как пользовательская функция в строке 160 с помощью оператора `DEF FN`.

Пользовательскую функцию можно определять, используя и символьные функции.

Пример 11.13. Определение длины слов

Требуется разработать программу определения и вывода длины вводимых слов.

Программа 11.13

```
'11DEFfn$.bas
10 CLS
20 DIM sl$(5)
30 DEF fns$ (a$) = STR$(LEN(a$))
40 FOR i = 1 TO 5
50 DATA ВНУЧКА,НЕВЕСТКА,БАБУШКА,ДЕДУШКА,СНОХА
60 READ sl$(i)
70 PRINT fns$(sl$(i)) + " букв содержит слово " + sl$(i)
80 NEXT i
90 END
```

Результат работы программы 11.13 приведен на рис. 11.15.

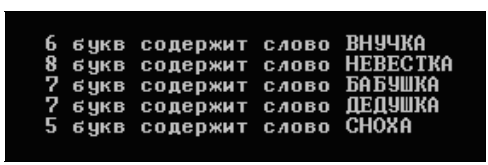


Рис. 11.15. Программа определения длины слов

Пользовательская функция в строке 30 определяет длину слов (оператор `LEN`), вводимых посредством пары операторов `READ` и `DATA`, а затем делает эту длину строковой (оператор `STR$`), чтобы обеспечить вывод результата оператором `PRINT` в строке 70.

ГЛАВА 12

Программирование игр

В настоящей главе рассмотрим разработку программы несложных игр.

Пусть следует разработать программу, реализующую пасьянс "Пирамида", относящийся к пасьянсам вероятностного типа, при перекладке которого конечный результат — сойдется или не сойдется пасьянс — зависит лишь от первоначального распределения карт. Для раскладки нужна колода в 36 карт. Выкладываются 9 кучек карт, по 3 в каждой, картинкой вниз. Последние 9 карт кладутся поверх каждой кучки картинкой вверх.

Парные карты удаляются. Освободившаяся верхняя карта кучки переворачивается картинкой вверх. Парные карты снова изымаются и т. д. до тех пор, пока все кучки не будут разобраны. 9 кучек можно расположить пирамидкой. Место, освободившееся в ряду после удаления четвертой, самой нижней карты в кучке, не возобновляется, количество открытых карт, таким образом, уменьшается, и выбирать приходится уже из меньшего количества карт. Естественно поэтому, что если в числе открытых карт окажутся 3 карты одинакового достоинства, то следует выбросить те две, под которыми еще есть закрытые карты. Пасьянс легкий и рассчитан в основном на внимание.

Предлагаемый вариант интерфейса программы приведен на рис. 12.1.

Логическую структуру создаваемой программы определяет блок-схема, показанная на рис. 12.2. Отметим, что большинство блоков блок-схемы являются укрупненными, т. е. представляющими определенную структуру. Например, блоки 1, 2, 3 детализируются циклическими структурами. Однако авторы сознательно пошли на эту укрупненность, поскольку в более подробной блок-схеме утонула бы логика программы.

Начинать разработку программы лучше с рисования колоды карт. При этом есть два пути. Первый путь — использовать имеющиеся значки мастей, но тогда позиция карты будет привязана к сетке экрана. Второй путь — нарисовать свои значки, например с помощью оператора `DRAW`. Создатели используемой в данной программе колоды карт — студентки СПбГУСЭ Юлия Соколова, Маргарита Христофорова и Любовь Гусарова — избрали первый путь.

Для начала следует разработать самую габаритную карту — 10-ку, после чего можно будет определить размеры рамки карты и произвести разметку расклада карт на экране. Тогда начальный фрагмент программы для разработки колоды карт будет выглядеть следующим образом (программа 12.1).

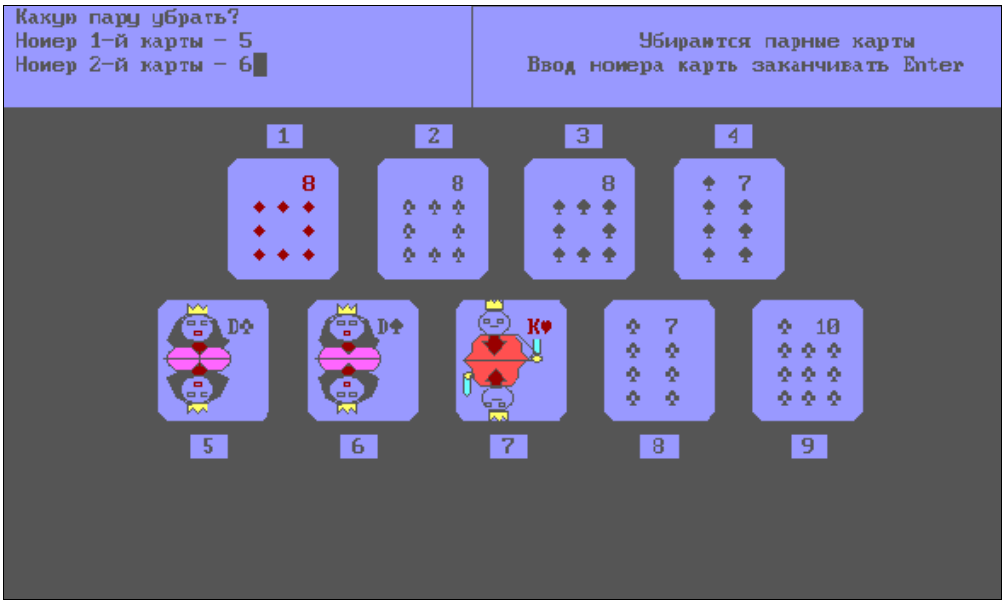


Рис. 12.1. Интерфейс программы "Пасьянс Пирамида"

Программа 12.1

```
'32Pirmda.bas      Пасьянс Пирамида
10 CLS
20 SCREEN 9                    'установка экранного режима
30 COLOR 8, 15                 'установка экранных цветов
40 RANDOMISE 32767 – TIMER
50 DIM kart1(5, 9), kart2(36)
'================================================================
60 j = 1
70 zn$ = CHR$(3)
80 my = 14
90 mx = 17 + 12 * (j – 1)
100 GOSUB 5
GOTO 900
'======== Подпрограмма рисования 10-к ============================
5 LOCATE my, mx
PRINT 10
FOR k1 = my + 1 TO my + 3
FOR k2 = mx – 2 TO mx + 3  STEP 2
LOCATE k1, k2
PRINT zn$
NEXT k2, k1
LOCATE my, mx – 2
PRINT zn$
RETURN
'================================================================
900 END
```

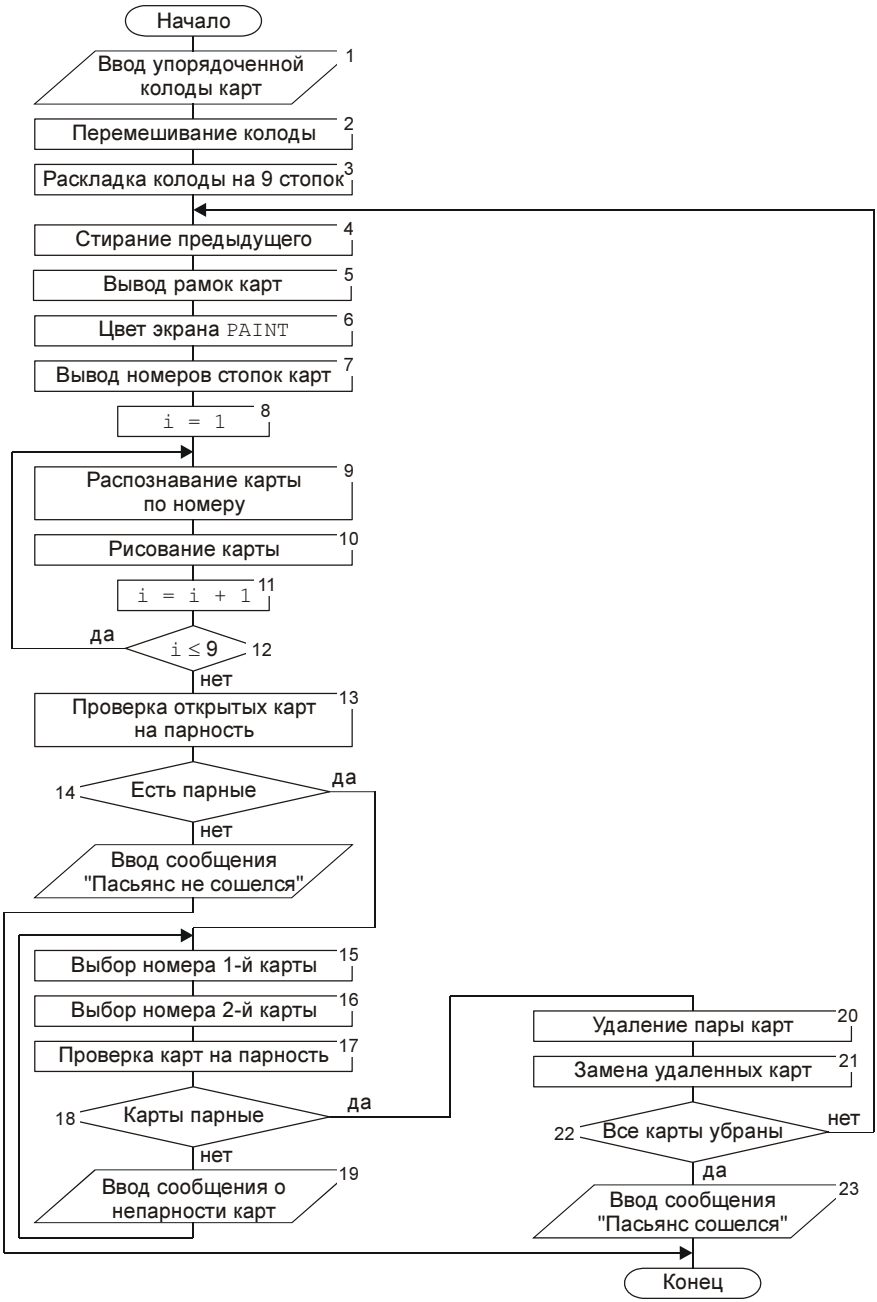


Рис. 12.2. Блок-схема программы "Пасьянс Пирамида"

Подпрограмму рисования 10-ки комментировать не станем. При желании читатель может разобрать ее самостоятельно. Для остальной части приведенного фрагмента программы дадим пояснения.

Оператор `RANDOMISE` в строке 40 запускает генератор случайных чисел. В строке 50 оператор `DIM` задает два массива: `kart2` — для записи номеров карт в линию — от 1 до 36 (табл. 12.1),

а kart1 — для карт, разложенных на 9 кучек (стопок). Номера 1—9 соответствуют масти червей, 10—18 — масти бубен, 19—27 — масти треф и 28—36 — пик. Таким образом, номер 1 — это 6-ка червей, 2 — 7-ка червей и т. п., 9 — туз червей. Номер 10 — 6-ка бубен, 11 — 7-ка бубен, 12 — 8-ка бубен и т. п., 18 — туз бубен. Номер 19 — 6-ка треф и т. п., 36 — туз пик.

Таблица 12.1. *Таблица соответствия присвоенных номеров картам*

	1	2	3	4	5	6	7	8	9
Черви	6-ка	7-ка	8-ка	9-ка	10-ка	Валет	Дама	Король	Туз
	10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
Бубны	6-ка	7-ка	8-ка	9-ка	10-ка	Валет	Дама	Король	Туз
	19	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**
Трефы	6-ка	7-ка	8-ка	9-ка	10-ка	Валет	Дама	Король	Туз
	28	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
Пики	6-ка	7-ка	8-ка	9-ка	10-ка	Валет	Дама	Король	Туз

Перед тем как определить положение рамок карт, следует задать поле вывода сообщений и вывода подсказки. Поскольку при рисовании карт используются значки мастей, имеющиеся в таблице ASCII, то и поля карт, как и поля для вывода сообщений и подсказки, лучше всего оставить цветом фона, задаваемым оператором COLOR в строке 30, а остальной экран закрасить другим цветом, что и делает оператор PAINT в строке 650 (программа 12.2).

Для определения размеров рамок карт и их расположения на экране создается временный цикл (потом он убирается из программы) FOR...NEXT, выводящий 9 десяток — 4 в верхнем ряду и 5 в нижнем. Затем с помощью оператора DRAW рисуется рамка (строка 625 в программе 12.2), которую необходимо проверить и, если надо, скорректировать на валетах, дамах и королях.

Блок рисования рамок карт, оформляемый затем как подпрограмма, приведен в программе 12.2.

Программа 12.2

```
'==5===== Подпрограмма рисования рамок карт ========================
600 xkN = 170
605 xkN = xkN + 324
610 FOR yk = 100 TO 183 STEP 83
615 FOR xk = xkN TO xkK STEP 95
620 PSET (xk — 27, yk — 5), 0
625 DRAW "c8 e6 r60 f6 d60 g6 l60 h6 u60"          'рамка
630 NEXT xk
635 xkN = xkN — 44
640 xkK = xkK + 50
645 NEXT yk
650 PAINT (1, 120), 2, 8                  'закрашивает фон
```

Причем в строках 600—605 находятся начальные установки по xk для верхнего ряда карт, которые затем переустанавливаются в строках 635—640 для нижнего ряда карт. Заметим,

что начальные установки выражены через xkN в строке 600 с тем, чтобы, меняя только значение xkN в строке 600, можно было бы передвигать без изменения всю конструкцию из 9 рамок. Для вывода рамок организуется двойной цикл по yk и xk в строках 610—645, в теле которого находятся рисующий рамку оператор DRAW в строке 625 и оператор PSET в строке 620, определяющий положение рамок. Начальное и конечное значения в цикле по yk, а также значения STEP в обоих циклах легче подобрать опытным путем. После вывода рамок карт экран закрашивается оператором PAINT. Поскольку для обновления после хода карты стираются выводимым оператором LINE прямоугольником цвета фона из оператора COLOR в строке 30, то координаты в операторе PAINT выбираются таким образом, чтобы точка начала закраски находилась внутри этого прямоугольника.

Завершается эта подпрограмма блоком рисования номеров стопок карт в строках 655—680, для чего организуется цикл FOR...NEXT (программа 12.3).

Программа 12.3

```
'==6===== Подпрограмма рисования номеров стопок карт ===============
655 FOR k =1 TO 9
660 IF k < 5 THEN ykP = 6: xkPo = 10 ELSE ykP = 19: xkPo = -44
665 xkP = xkPo + k * 12
670 LOCATE ykP, xkP
675 PRINT k
680 NEXT k
685 RETURN
```

В строке 670 стоит оператор LOCATE, входящий в тело цикла и задающий позицию вывода очередного номера оператором PRINT в строке 675. Строки 660—665 определяют начальные установки и шаг для параметров оператора LOCATE, в зависимости от того, в каком ряду необходимо вывести номер. Эти значения подбираются, согласуясь с положением рамок карт, и наоборот — положения рамок карт согласуются с выведенными номерами стопок.

Итак, рассмотрены блоки, определяющие интерфейс программы. Теперь обратимся к структуре входных данных. Для этого в строке 50 оператором DIM объявляются два массива — двумерный kart1(5, 9) и одномерный kart2(36). В kart2(36) записывается неперемешенная колода (в соответствии с табл. 12.1), а в kart1(5, 9) — перемешенная и разбитая на 9 стопок. Структура массива kart1(5, 9) представлена в табл. 12.2, где kart1(5, i) — все открытые карты. Если в ячейку записан 0, то значит там карты нет.

Таблица 12.2. *Структура массива* `kart1(5, 9)`

		i								
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
j	**1**	0	0	0	0	0	0	0	0	0
	2	9	5	15	17	4	35	14	11	7
	3	26	27	2	13	1	31	22	32	16
	4	19	6	33	24	10	36	3	28	18
	5	12	21	30	29	25	34	8	20	23

Строка kart1(1, i) — техническая, необходимая для того, чтобы получать для удаленной карты значение 0 при перезаписи. Поясним, как будут изменяться данные по мере удаления

парных карт. Пусть во 2-й стопке осталась одна карта, а в стопке 7 — три карты, тогда будем иметь следующие значения для элементов массива:

```
kart1(1, 2) = 0          kart1(1, 7) = 0
kart1(2, 2) = 0          kart1(2, 7) = 0
kart1(3, 2) = 0          kart1(3, 7) = 14
kart1(4, 2) = 0          kart1(4, 7) = 22
kart1(5, 2) = 5          kart1(5, 7) = 3
```

Заполнение массива `kart2(36)` осуществляется посредством оператора цикла `FOR...NEXT` в строках 120—140.

```
120 FOR i1 = 1 TO 36       'заполнение массива
130 kart2(i1) = i1
140 NEXT i1
```

Теперь рассмотрим процедуру перемешивания колоды карт и раскладывания ее на 9 стопок (кучек). Принцип перемешивания проиллюстрирован на рис. 12.3, который показывает первых четыре шага процедуры. Далее приведен соответствующий фрагмент программы:

```
150 nk1 = 36
160 FOR i2 = 2 TO 5        'двойной цикл перемешивания и
170 FOR i3 = 1 TO 9        'раздачи карт
180 i4 = INT(1 + nk1 * RND(1))
190 kart1(i2, i3) = kart2(i4)
200 SWAP kart2(i4) = kart2(nk1)
210 nk1 = nk1 - 1
220 NEXT i3, i2
```

Организуется двойной цикл `FOR...NEXT`, в теле которого в строке 180 с помощью генератора случайных чисел выбирается номер карты. На первом шаге процедуры, как видно из рис. 12.3, *а*, выбор осуществляется из 36 карт, поскольку в строке 150 начальное значение множителя `nk1` при `RND` присваивается равным 36. Выбранная карта кладется вниз в 1-ю стопку — в строке 190 `kart1(2, 1) = 9` (см. табл. 12.2). Затем оператор `SWAP` в строке 200 меняет в массиве `kart2(36)` местами элементы под номером 9 и 36. В строке 210 значение множителя при `RND` уменьшается на 1 и становится равным 35. Поэтому выпавшая на первом шаге и перемещенная на позицию 36 в массиве `kart2(36)` карта выпадает из выбора на втором шаге, что видно из рис. 12.3, *б*. На втором шаге процедуры случайным образом выбирается нижняя карта 2-й стопки (`kart1(2, 2) = 5` — см. табл. 12.2). Продолжая процедуру таким образом (рис. 12.3, *в* и *г*), перемешиваем карты и раскладываем их на 9 стопок по 4 карты. Поскольку во вспомогательном ряду `kart1(1, i)` должны быть записаны нули, то начальное значение счетчика цикла `i2` в строке 160 принимается равным 2.

Центральным блоком создаваемой программы является блок игры (программа 12.4).

Программа 12.4

```
'==3===== Блок игры ============================================
230 s = 1
240 DO
250 LINE (98, 88) -STEP(460, 160), 0, BF
260 GOSUB 600                    'подпрограмма рисования рамок карт
270 IF s = 0 THEN e = 1: GOTO 290
280 GOSUB 570                          'подпрограмма выбора карт
```

```
290 COLOR 8
300 LINE (0, 0) -STEP(299, 59), 8, BF    'очищает поле для сообщений
310 IF e = 1 THEN LOCATE 2, 4: PRINT "Пасьянс сошелся": EXIT DO
320 IF e = 2 THEN LOCATE 2, 4: PRINT "Пасьянс не сошелся": EXIT DO
    '---- Проверка на парность -------------------------------------
330 v = 0
340 FOR m1 = 1 TO 8
350 FOR m2 = m1 TO 8
360 IF kart1(5, m1) > 0 AND kart1(5, m2 + 1) > 0 THEN  ELSE 390
370 n1n = ABS (kart1(5, m1) - kart1(5, m2 + 1))
380 IF n1n MOD 9 = 0 THEN v = 1: EXIT FOR
390 NEXT m2, m1
400 IF v = 0 THEN e = 2: GOTO 290
    '---- Вывод сообщений -------------------------------------
410 LOCATE 1, 2
420 PRINT "Какую пару убрать?"
430 LOCATE 2, 2
440 INPUT "Номер 1-й карты - ", n1
450 LOCATE 3, 2
460 PRINT "Номер 2-й карты - ", n2
    '---- Вывод сообщений -----------------------------------------
470 nn = ABS (kart1(5, m1) - kart1(5, n2))
480 IF nn MOD 9 = 0 THEN 530
490 LOCATE 4, 2: PRINT "Это не пара": SLEEP 2: GOTO 290
    '-------------------------------------------------------------
500 nn1 = n1: GOSUB 700           'Подпрограмма открытия новых карт
510 nn1 = n2: GOSUB 700
    '---- Условие окончания игры ----------------------------------
520 s = 0
530 FOR i = 1 TO 9
540 s = s + kart1(5, i)
550 NEXT i
560 LOOP
```

Блок игры представляет собой цикл `DO...LOOP`, находящийся в строках 240—560. Пасьянс завершается, когда убраны все карты (`"Пасьянс сошелся"`), либо когда среди открытых карт нет парных (`"Пасьянс не сошелся"`). Поэтому в блоке предусмотрены 2 проверки. Для проверки на парность организован двойной (вложенный) цикл `FOR...NEXT` в строках 340—390. Он реализует последовательную проверку на парность открытой карты 1-й кучки с открытыми картами 2-й, 3-й, ... , 9-й кучек, затем открытой карты 2-й кучки с открытыми картами 3-й, 4-й, ... , 9-й кучек и т. д. Проверка имеет смысл, если в проверяемых кучках есть открытые карты, т. е. вообще имеются карты. Если карты нет, то `kart1(5, m1) = 0`. Поэтому контроль наличия карт осуществляется в строке 360. Поскольку масть содержит 9 карт, то карты будут парными, если разность их значений, взятая по абсолютной величине, окажется кратной 9. Такая проверка производится в строках 370—380. Результат проверки на парность обрабатывается в строке 400. Если есть парные карты, то переключатель `v` в строке 380 получает значение 1, и программа проходит строку 400. Если же парных карт нет, то переключатель `v` остается равным 0. Тогда переключатель `e` в строке 400 получает значение 2, а управление передается на окончание игры с сообщением `"Пасьянс не сошелся"`. Соответствующая картинка представлена на рис. 12.4.

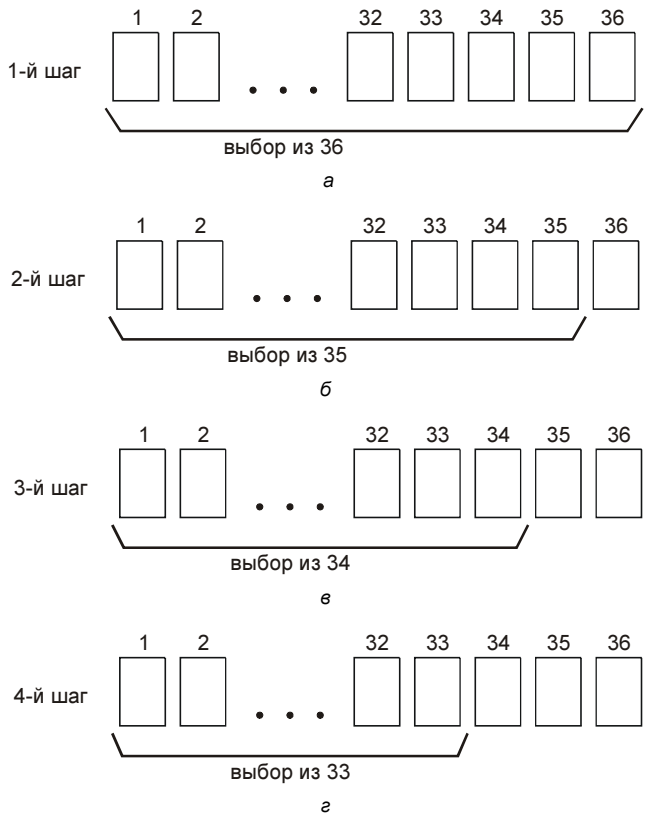


Рис. 12.3. Пояснение процедуры перемешивания карт

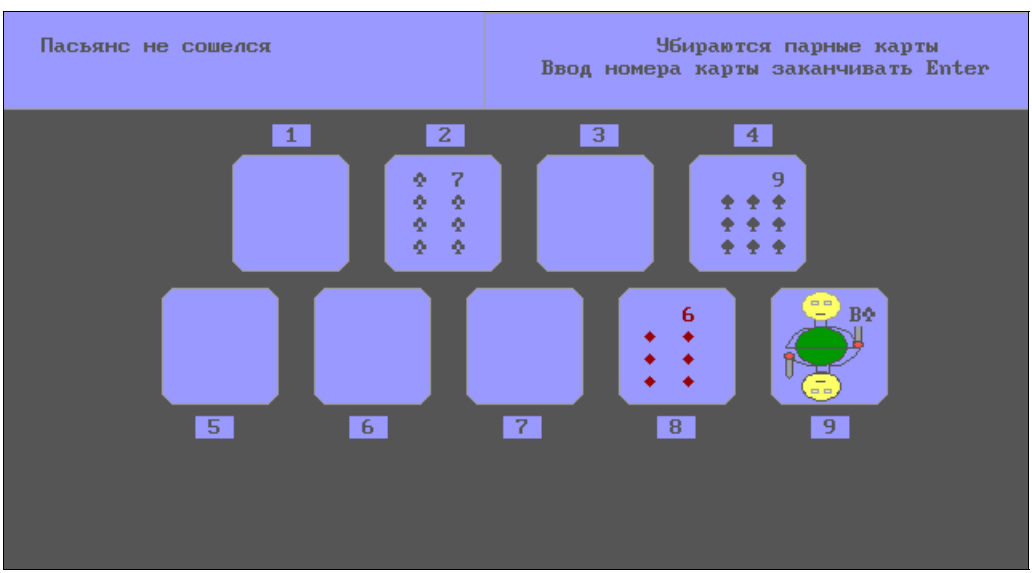


Рис. 12.4. Окончание игры с сообщением "Пасьянс не сошелся"

Вторая проверка на окончание производится в строках 530—560. Если все карты убраны, то сумма всех 9-ти `kart1(5, m1)` будет равна 0. При выполнении этого условия переключателю `e` в строке 270 присваивается значение, равное 1, и управление передается на окончание игры с сообщением "`Пасьянс сошелся`". Вывод сообщений перед окончанием игры обеспечивается операторами в строках 290—320.

Итак, оператор `LINE` в строке 250 стирает предыдущее. Строка 260 обеспечивает вывод рамок карт, и, если нет команды на окончание пасьянса в строке 270, рамки заполняются соответствующими картами — строка 280. Затем, при положительной проверке на парность программа переходит к выбору убираемой пары карт, что осуществляется в строках 410—460. Выбранная пара карт также проходит проверку на парность с помощью операторов в строках 470—490. Если выбранные карты не парные, то появляется сообщение об этом, после чего выбор следует повторить. Если же проверка на парность пройдена успешно, то выбранные карты заменяются на новые с помощью подпрограммы открытия новых карт — строки 500—510.

Подпрограмма открытия новых карт осуществляет для соответствующей кучки перезапись значения `kart1(1, m1)` **в** `kart1(2, m1)`, `kart1(2, m1)` **в** `kart1(3, m1)`, `kart1(3, m1)` **в** `kart1(4, m1)`, `kart1(4, m1)` **в** `kart1(5, m1)`. Далее приведен этот фрагмент:

```
'==7===== Подпрограмма открытия новых карт ====================
700 FOR k =1 TO 4
710 kart1(6 — k, nn1) = kart1(6 — k — 1, nn1)
720 NEXT k
730 RETURN
'============================================================
```

Как видно, это цикл `FOR...NEXT` с конечным значением счетчика цикла, равным числу карт в кучке, в теле которого и осуществляется указанная перезапись.

Остается теперь рассмотреть подпрограмму выбора карты. Соответствующий фрагмент приведен в программе 12.5.

Программа 12.5

```
'==4===== Подпрограмма выбора карты  ==============================
570 FOR j = 1 TO 9
573 mast = kart1(5, j)
       SELECT CASE j              'выбор начальных установок для карт
CASE 1 TO 4: my = 8: Mo = 23: dj = 1
               Y = 100: Xo = 170
               ykF = 8: xkFo = 25
CASE 5 TO 9: my = 14: Mo = 17: dj = 5
               Y = 183: Xo = 126
               ykF = 14: xkFo = 19
       END SELECT
576 mx = Mo + 12 * (j — dj)       '6 — 10-ки
579 x = Xo + 96 * (j — dj)        'картинки
582 xkF = xkFo + 12 * (j — dj)    'значки картинок
       SELECT CASE mast           'выбор масти (значка и цвета)
CASE 0: GOTO 590
CASE 1 TO 9:     zn$ = CHR$(3): COLOR 4     'черви
CASE 10 TO 18: zn$ = CHR$(4): COLOR 4       'бубны
```

```
CASE 19 TO 27: zn$ = CHR$(5): COLOR 8      'кресты
CASE 28 TO 36: zn$ = CHR$(6): COLOR 8      'пики
       END SELECT
       SELECT CASE mast            'выбор вида (достоинства) карты
CASE 1, 10, 19, 28: nk = 1         '6-ки
CASE 2, 11, 20, 29: nk = 2         '7-ки
CASE 3, 12, 21, 30: nk = 3         '8-ки
CASE 4, 13, 22, 31: nk = 4         '9-ки
CASE 5, 14, 23, 32: nk = 5         '10-ки
CASE 6, 15, 24, 33: nk = 6         'вальты
CASE 7, 16, 25, 34: nk = 7         'дамы
CASE 8, 17, 26, 35: nk = 8         'короли
CASE 9, 18, 27, 36: nk = 9         'тузы
       END SELECT
585 ON nk GOSUB 1, 2, 3, 4, 5, 6, 7, 8, 9   'выбор подпрограммы карты
590 NEXT j
RETURN
```

Подпрограмма выбора карты представляет собой цикл `FOR...NEXT` с конечным значением счетчика цикла, равным числу кучек, в теле которого расположены 3 оператора `SELECT CASE`. Первый оператор `SELECT CASE` отвечает за параметры, передаваемые в подпрограммы рисования карт. Эти параметры задаются формулами в строках 576—582, а оператор `SELECT CASE` реализует переменную часть — в `CASE 1 TO 4` значения для карт верхнего ряда и в `CASE 5 TO 9` значения для карт нижнего ряда. Второй оператор `SELECT CASE` определяет значок масти и его цвет. Третий оператор `SELECT CASE` осуществляет выбор вида (достоинства) карты, задавая значение параметра `nk` в операторе `ON...GOSUB` в строке 585. В зависимости от номера карты определяется значение параметра `nk`, а оператор `ON...GOSUB` передает управление в соответствующую подпрограмму рисования карты. Например, для `kart1(5, m1) = 17` управление передается в подпрограмму рисования королей.

Полная версия разрабатываемой программы приведена в программе 12.6.

Программа 12.6

```
'32Pirmda.bas      Пасьянс Пирамида
10 CLS
20 SCREEN 9
30 COLOR 8, 15
40 RANDOMISE 32767 - TIMER
50 DIM kart1(5, 9), kart2(36)
'==1===== Блок вывода подсказки ===================================
60 LINE (0, 0) -STEP(650, 60), 8, B
70 LINE (0, 0) -STEP(300, 60), 8, B      'рамка поля
80 LOCATE 2, 52
90 PRINT "Убираются парные карты"
100 LOCATE 3, 43
110 PRINT "Ввод номера карты заканчивать Enter"
'==2===== Блок заполнения массива и перемешивания колоды ===========
120 FOR i1 = 1 TO 36                     'заполнение массива
130 kart2(i1) = i1
140 NEXT i1
```

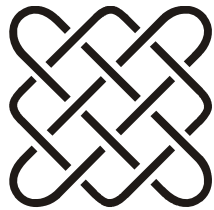
```
     '-------------------------------------------------------------
150 nk1 = 36
160 FOR i2 = 2 TO 5                        'двойной цикл перемешивания и
170 FOR i3 = 1 TO 9                        'раздачи карт
180 i4 = INT(1 + nk1 * RND(1))
190 kart1(i2, i3) = kart2(i4)
200 SWAP kart2(i4) = kart2(nk1)
210 nk1 = nk1 - 1
220 NEXT i3, i2
'==3===== Блок игры ==============================================
230 s = 1
240 DO
250 LINE (98, 88) -STEP(460, 160), 0, BF
260 GOSUB 600                       'подпрограмма рисования рамок карт
270 IF s = 0 THEN e = 1: GOTO 290
280 GOSUB 570                              'подпрограмма выбора карт
290 COLOR 8
300 LINE (0, 0) -STEP(299, 59), 8, BF     'очищает поле для сообщений
310 IF e = 1 THEN LOCATE 2, 4: PRINT "Пасьянс сошелся": EXIT DO
320 IF e = 1 THEN LOCATE 2, 4: PRINT "Пасьянс не сошелся": EXIT DO
     '---- Проверка на парность ------------------------------------
330 v = 0
340 FOR m1 = 1 TO 8
350 FOR m2 = m1 TO 8
360 IF kart1(5, m1) > 0 AND kart1(5, m2 + 1) > 0 THEN  ELSE 390
370 n1n = ABS (kart1(5, m1) - kart1(5, m2 + 1))
380 IF n1n MOD 9 = 0 THEN v = 1: EXIT FOR
390 NEXT m2, m1
400 IF v = 0 THEN e = 2: GOTO 290
     '---- Вывод сообщений -----------------------------------------
410 LOCATE 1, 2
420 PRINT "Какую пару убрать?"
430 LOCATE 2, 2
440 INPUT "Номер 1-й карты - ", n1
450 LOCATE 3, 2
460 PRINT "Номер 2-й карты -", n2
     '---- Вывод сообщений -----------------------------------------
470 nn = ABS (kart1(5, m1) - kart1(5, n2))
480 IF nn MOD 9 = 0 THEN 530
490 LOCATE 4, 2: PRINT "Это не пара": SLEEP 2: GOTO 290
     '-------------------------------------------------------------
500 nn1 = n1: GOSUB 700           'Подпрограмма открытия новых карт
510 nn1 = n2: GOSUB 700
     '---- Условие окончания игры ----------------------------------
520 s = 0
530 FOR i = 1 TO 9
540 s = s + kart1(5, i)
550 NEXT i
560 LOOP
GOTO 900
'==4===== Подпрограмма выбора карты ===============================
570 FOR j = 1 TO 9
573 mast = kart1(5, j)
       SELECT CASE j              'выбор начальных установок для карт
```

```
CASE 1 TO 4: my = 8: Mo = 23: dj = 1
             Y = 100: Xo = 170
             ykF = 8: xkFo = 25
CASE 5 TO 9: my = 14: Mo = 17: dj = 5
             Y = 183: Xo = 126
             ykF = 14: xkFo = 19
      END SELECT
576 mx = Mo + 12 * (j — dj)          '6 — 10-ки
579 x = Xo + 96 * (j — dj)           'картинки
582 xkF = xkFo + 12 * (j — dj)       'значки картинок
      SELECT CASE mast               'выбор масти (значка и цвета)
CASE 0: GOTO 590
CASE 1 TO 9:     zn$ = CHR$(3): COLOR 4   'черви
CASE 10 TO 18: zn$ = CHR$(4): COLOR 4     'бубны
CASE 19 TO 27: zn$ = CHR$(5): COLOR 8     'кресты
CASE 28 TO 36: zn$ = CHR$(6): COLOR 8     'пики
      END SELECT
      SELECT CASE mast             'выбор вида (достоинства) карты
CASE 1, 10, 19, 28: nk = 1         '6-ки
CASE 2, 11, 20, 29: nk = 1         '7-ки
CASE 3, 12, 21, 30: nk = 1         '8-ки
CASE 4, 13, 22, 31: nk = 1         '9-ки
CASE 5, 14, 23, 32: nk = 1         '10-ки
CASE 6, 15, 24, 33: nk = 1         'вальты
CASE 7, 16, 25, 34: nk = 1         'дамы
CASE 8, 17, 26, 35: nk = 1         'короли
CASE 9, 18, 27, 36: nk = 1         'тузы
      END SELECT
585 ON nk GOSUB 1, 2, 3, 4, 5, 6, 7, 8, 9  'выбор подпрограммы карты
590 NEXT j
RETURN
'==5===== Подпрограмма рисования рамок карт ========================
600 xkN = 170
605 xkN = xkN + 324
610 FOR yk = 100 TO 183 STEP 83
615 FOR xk = xkN TO xkK STEP 95
620 PSET (xk — 27, yk — 5), 0
625 DRAW "c8 e6 r60 f6 d60 g6 l60 h6 u60"       'рамка
630 NEXT xk
635 xkN = xkN — 44
640 xkK = xkK + 50
645 NEXT yk
650 PAINT (1, 120), 2, 8                        'закрашивает фон
'==6===== Подпрограмма рисования номеров стопок карт ===============
655 FOR k =1 TO 9
660 IF k < 5 THEN ykP = 6: xkPo = 10 ELSE ykP = 19: xkPo = -44
665 xkP = xkPo + k * 12
670 LOCATE ykP, xkP
675 PRINT k
680 NEXT k
685 RETURN
```

```
'==7===== Подпрограмма открытия новых карт ========================
700 FOR k =1 TO 4
710 kart1(6 — k, nn1) = kart1(6 — k — 1, nn1)
720 NEXT k
730 RETURN
'==================================================================
900 END

'==8===== Подпрограмма рисования 6-к ==============================
1 LOCATE my, mx
PRINT 6
FOR k1 = my + 1 TO my + 3
FOR k2 = mx — 2 TO mx + 1  STEP 3
LOCATE k1, k2
PRINT zn$
NEXT k2, k1
RETURN
'==9===== Подпрограмма рисования 7-к ==============================
2 LOCATE my, mx
PRINT 7
FOR k1 = my + 1 TO my + 3
FOR k2 = mx — 2 TO mx + 1  STEP 3
LOCATE k1, k2
PRINT zn$
NEXT k2, k1
LOCATE my, mx — 2
PRINT zn$
RETURN
'==10==== Подпрограмма рисования 8-к ==============================
3 LOCATE my, mx
PRINT 8
FOR k1 = my + 1 TO my + 3
FOR k2 = mx — 2 TO mx + 3  STEP 4
LOCATE k1, k2
PRINT zn$
NEXT k2, k1
LOCATE my + 1, mx
PRINT zn$
LOCATE my + 3, mx
PRINT zn$
RETURN
'==11==== Подпрограмма рисования 9-к ==============================
4 LOCATE my, mx + 1
PRINT 9
FOR k1 = my + 1 TO my + 3
FOR k2 = mx — 2 TO mx + 2  STEP 2
LOCATE k1, k2
PRINT zn$
NEXT k2, k1
RETURN
```

```
'==12==== Подпрограмма рисования 10-к ==============================
5 LOCATE my, mx
PRINT 10
FOR k1 = my + 1 TO my + 3
FOR k2 = mx — 2 TO mx + 3  STEP 2
LOCATE k1, k2
PRINT zn$
NEXT k2, k1
LOCATE my, mx — 2
PRINT zn$
RETURN
'==14==== Подпрограмма рисования валетов ===========================
6 CIRCLE (x, y), 12, 7                              'верхняя голова
PAINT STEP(1, -1), 14, 7
CIRCLE STEP(-5, -1), 2, 7
CIRCLE STEP(8, 0), 2, 7
LINE STEP(-7, 49) -STEP(5, 0), 8
LINE STEP(-5, -42) -STEP(5, 0), 8
CIRCLE STEP(-2, 20), 16, 8
DRAW "bm-4,-16 c8 d4 g1 l6 g1 l1 g2 l1 g2 l1 g2 l1 g2 l1"
DRAW "r46 d1 g2 l1 g2 l1 g2 l1 g2 l1 g1 l7 d5 l8 u5 l6 h1"
DRAW "l2 h1 l2 h1 l1 h1 d10 g2 h2 u10 r5 bm-2,+10 p7,8"
DRAW "bm+22,-34 c8 d4 r6 f1 r2 f1 r2 f1 r1 f2 d1 f2 d1"
DRAW "u10 e2 f2 d10 l5 bm+3,-3 p7,8"
CIRCLE STEP(0, 5), 3, 8                             'верхний кулак
PAINT STEP(0, 0), 12, 8
CIRCLE STEP(-43, 7), 3, 8                           'нижний кулак
DRAW "bf1 p12,8 bm+23,-17 p2,8 bd20 p2,8"
CIRCLE STEP(-4, 15), 12, 8                          'нижняя голова
PAINT STEP(1, -1), 14, 8
CIRCLE STEP(-5, 4), 2, 7
CIRCLE STEP(8, 0), 2, 7
LOCATE ykF, xkF
PRINT "B"; zn$
RETURN
'==15==== Подпрограмма рисования дам ===============================
7 CIRCLE (x — 2, y + 6), 11, 8
CIRCLE STEP(4, -3), 2, 8          'глаз
CIRCLE STEP(-8, 0), 2, 8          'глаз
CIRCLE STEP(4, 6), 2, 4           'губы
CIRCLE STEP(-9, 15), 10, 8        'левая грудь
PAINT STEP(-1, 1), 13, 8
CIRCLE STEP(21, -1), 10, 8        'правая грудь
PAINT STEP(-1, 1), 13, 8
DRAW "bm-13,-12 c8 d2 g2"
DRAW "bm+9,-4 d2 f2 bm-5,+3 p4,8"                       'верхний галстук
DRAW "bm-7,-22 c8 u6 f3 e3 f3 e3 d6 l12 bm+5,-2 p14,8"  'верхняя корона
DRAW "bm-5,+2 c8 l1 g1 l1 g1 l1 d1 l1 d1 l1 d1 l1 d2 l1 d2 l1"
DRAW "d2 l1 d2 l1 d2 l1 r1 d1 r1 d1"
DRAW "r1 d1 r1 d1 bm+1,-5 p8,8"             'волосы
DRAW "bm+30,+6 c8 u1 r1 u1 r1 u1 r1 u1 r1 u1 l1 h1 l1 h1 u1 h1"
```

```
DRAW "u2 h1 u1 h1 u2 h1 l1 h1 l1 h1 l1 h1 bm+7,+10 p8,8"       'волосы
DRAW "bm-17,+43 c8 l1 h1 l1 h1 l1 h1 l1 h1  l1 h1 u1 h1 u1
DRAW "h1 u1 h1 u1 h1 u1 h1 u1 r1 e1 r1 e1 r1 e1 r1 u1"
DRAW "bm+18,+19 r1 e1 r1 e1 r1 e1 r1 e1 r1 u1 e1 u1"
DRAW "e1 u1 e1 u1 e1 u1 e1 l1 u1 l1 h1 l1 h1 l1 h1 l1"
LINE STEP(-33, -8) -STEP(42, 0), 8
DRAW "bm-24,+11 c8 u2 h2"
CIRCLE STEP(5, 11), 11, 8
DRAW "bm+4,-7 c8 u2 e2 bm-5,+2 p4,8"               'нижний галстук
CIRCLE STEP(3, 12), 2, 8                           'глаз
CIRCLE STEP(-8, 0), 2, 8                           'глаз
CIRCLE STEP(4, -6), 2, 8                           'губы
DRAW "bm-6,+11 c8 d6 e3 f3 e3 f3 u6 l12 bm+5,+2 p14,8"       'корона
DRAW "be15 p8,8 ml30 p8,8"    'нижние волосы
LOCATE ykF, xkF
PRINT "D"; zn$
RETURN
'==16==== Подпрограмма рисования королей ============================
8 PSET (x — 10, y — 5), 0
DRAW "c8 u6 f2 e2 f2 e2 f2 e2 d6 l12 bm+1,-1 p14,8"  'верхняя корона
DRAW "bm+3,+11 c8 r4 bm-5,+5 d2 g2 l3 f8 e8 l3 h2 u2 r10"
DRAW "f5 d5 f5 l42 e5 u5 e5 r25 f10 g10 d5 g5 l20 h5 u5"
DRAW "h5 bm+44,-4 u10 e2 f2 d10 l3 bm+1,-5 p11,8"    'верхний меч
PAINT STEP(-13, 4), 12, 8
PAINT STEP(-15, -4), 4, 8
CIRCLE STEP(3, 32), 10, 8              'нижняя голова
PAINT STEP(0, -11), 12, 8
CIRCLE STEP(25, -12), 3, 8             'верхний кулак
PAINT STEP(1, 1), 14, 8
CIRCLE STEP(-28, -22), 10, 8           'верхняя голова
CIRCLE STEP(-5, -1), 2, 8              'глаз
CIRCLE STEP(9, 0), 2, 8                'глаз
CIRCLE STEP(-8, 49), 2, 8              'глаз
CIRCLE STEP(9, 0), 2, 8                'глаз
CIRCLE STEP(-21, -17), 3, 8            'нижний кулак
PAINT STEP(0, 0), 14, 8
DRAW "bm+1,+0 c8 d10 g2 h2 u10 bm+1,+2 p11,8"        'верхний меч
DRAW "bm+15,+19 c8 d6 e3 f3 e3 f3"
DRAW "u6 l12 bm+6,+3 p14,8"            'нижняя корона
LINE STEP(-4, -10) -STEP(5, 0), 8      'губы
DRAW "c8 bu7 l6 u3 h1 l3 e7 f7 l3 g1 nd3 bl3 p4,8"   'нижний галстук
LOCATE ykF, xkF
PRINT "K"; zn$
RETURN
'==17==== Подпрограмма рисования тузов  ============================
9 LOCATE ykF, xkF
PRINT "T"; zn$
LOCATE ykF + 1, xkF — 2
PRINT zn$
RETURN
'====================================================================
```



ГЛАВА 13

Программирование музыки

Как не удивительно, но с помощью языка QBASIC можно создавать и музыкальные произведения. Итак, всего имеется семь нот. Вы, верно, слышали их названия. Это до, ре, ми и т. д. Если вы до сих пор их не знаете, можете и не заучивать, не мучить себя зря — это совершенно необязательно. Важно лишь, что каждая нота соответствует определенной частоте музыкального диапазона, который, заметим, несколько меньше, чем указанный выше звуковой диапазон, поскольку не все существующие звуки используются в музыке. Однако диапазон все же довольно велик, а вот нот всего семь, поэтому изобретателям нотной грамоты пришлось разбить его на девять частей, которые называются *октавами.* На практике чаще используется всего семь октав: они нумеруются цифрами от 0 до 6 (о них пойдет речь при полном описании процедуры `PLAY`).

Каждая октава задает непрерывный диапазон частот и содержит свои семь нот, а затем дробится ими на семь дискретных тонов (звуков). Таким образом, звук ноты зависит не только от ее названия, но еще и от октавы, в которой она содержится. Всего же различают 85—92 музыкальных звуков.

Октавы группируются в три *звуковых регистра*: низкий, средний и высокий. К *низкому регистру*, его еще называют басовым, относятся звуки субконтроктавы, контроктавы и большой октавы (последние две октавы для процедуры `PLAY` имеют номера 0 и 1).

Среднему регистру, он совпадает с диапазоном человеческого голоса, принадлежат звуки малой, первой и частично второй октавы (или номера 2, 3 и 4 для процедуры `PLAY`).

Высокий регистр включает третью, четвертую и пятую октавы (для программистов третья и четвертая октавы имеют номера 5 и 6). Среди музыкальных инструментов только орган и фортепиано включают весь диапазон музыкальной системы, а, например, контрабас и скрипка "работают" по краям, виолончель — посередине этого диапазона.

Разбив весь звуковой спектр примерно на 60 дискретных частот ($7 \times 9 = 63$, 7 нот, 9 октав, одна октава является неполной), мэтры сольфеджио обнаружили, что для всех имеющихся звуков обозначения все равно катастрофически не хватает. Поэтому они придумали всякие значки *альтерации*, в совокупности с которыми нота повышала или понижала частоту своего звучания.

Знак # (*диез*) повышает, а **b** (*бемоль*) понижает звук на полтона. Для повышения или понижения звука на целый тон используются, соответственно, знаки **x** (*дубль-диез*) и **bb** (*дубль-бемоль*). Для отмены знаков альтерации ставится знак **q** (*бекар*). Все эти знаки пишутся перед длительностью.

На бумаге ноты записываются при помощи *нотного стана* — пяти параллельных отрезков, равно отдаленных друг от друга, нумерации используются для каждого отрезка, начиная с нижнего. Бумага, разлинованная такими отрезками, называется нотной. На рис. 13.1 показано только две октавы: нижние и верхние ноты. Каждая из этих нот имеет определенную частоту звучания. Заметим только, что часто для расположения нот применяют дополнительные отрезки линий (расположенные горизонтально посередине ноты и под или над ней), как ниже основного стана, так и выше него. Это обусловлено тем, что пяти линий нотного стана недостаточно, чтобы записать все ноты выбранной мелодии. Теоретически, число таких дополнительных отрезков не ограничивается, но на практике обычно используется до пяти отрезков вниз и вверх. Но даже в этих редких случаях читать такие ноты чрезвычайно неудобно.

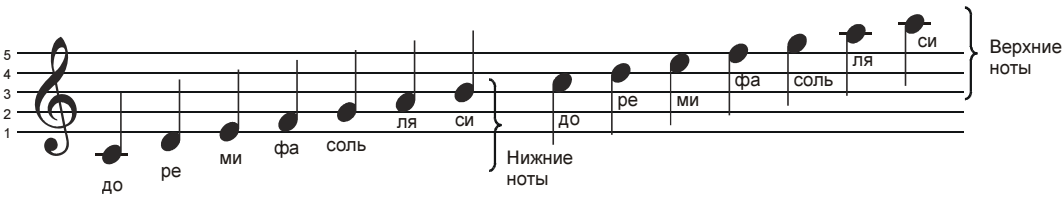


Рис. 13.1. Нижние и верхние ноты ключа соль

Поскольку имеются 9 октав, положение ноты на нотном стане не определяет ее *частоту* и ее *тон*. Нужно знать еще и имя октавы, в котором расположен этот нотный стан. Проблема решается достаточно оригинально: на нотном стане ставятся специальные знаки или *ключи.* Такими ключами начинается каждая строка произведения. Ключ записывается на одной из линий нотного стана и закрепляет за ней звук определенной высоты, от которого выполняется отсчет всех остальных звуков.

Чаще всего используется *ключ соль* (его еще называют скрипичным ключом), а также *басовый ключ фа*. Эти ключи начинают изображать, соответственно, на второй и четвертой линиях нотного стана (рис. 13.2), и поэтому говорят, что они расположены на этих линиях. Ключи получили свое название от нот, которые привязаны к соответствующим линиям. С использованием скрипичного ключа удобно записывать ноты среднего и высокого регистров, не только звуки скрипки, но и многих других инструментов. На рис. 13.2 индексами у названия нот отмечены имена октав.

Очень важным моментом в музыкальных произведениях является не только частота, но и *длительность* звучания ноты. Продолжительность звучания ноты измеряется частями интервала (иногда его называют тактом). Интервал — это длительность времени около секунды. Если говорить более точно, длительность интервала определяется количеством тактов, заданных у размера, исполняемых в секунду (что такое такты заданного размера, будет рассказано чуть позже). Но сейчас уже можно сказать, что количество таких тактов в минуту задается метрономом, а практикующий маэстро держит эту штуковину в голове. Ясно, что изменение единицы измерения длительности нот — уменьшение или увеличение интервала (такта) соответственно — *ускоряет* или *замедляет* мелодию. В табл. 13.1 показано, как выглядят ноты с различной длительностью звучания.

Если справа от головки ноты стоит *точка*, она увеличивает на половину длительность звучания этой ноты. Использование *двух точек* еще больше увеличивает длительность звучания: первая точка увеличивает ее на половину исходной длительности, а вторая — на половину той общей длительности, что получилась после действия первой точки. Например, точка после целой ноты делает продолжительность звучания равной 3/2, две точки после

целой ноты приводят к длительности звучания 9/4. Три и четыре точки применяются очень редко.

Таблица 13.1. *Длительность ноты*

Длительность ноты, интервал	Изображение	Команда длительности
1		`L1`
1/2		`L2`
1/4		`L4`
1/8		`L8`
1/16		`L16`
1/32		`L32`
1/64		`L64`

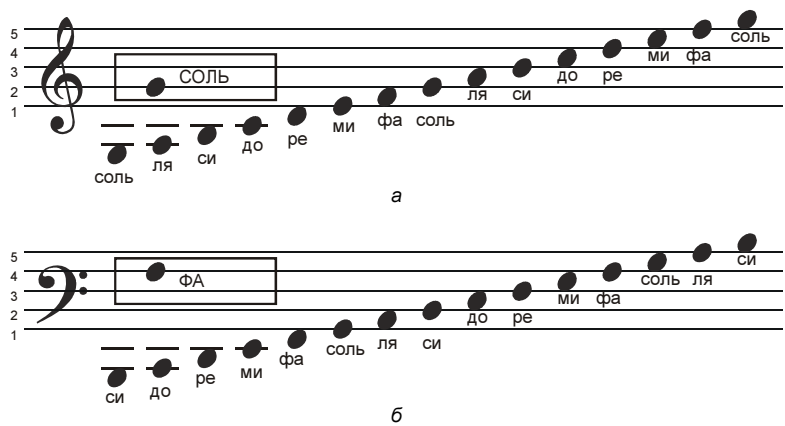


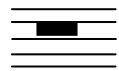
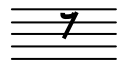
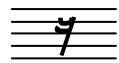
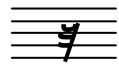
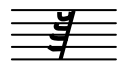
Рис. 13.2. Ключи соль (*а*) и фа (*б*)

В общем, вид ноты и ее положение относительно отрезков нотного стана — не что иное, как своеобразная система координат, которая предписывает каждой ноте вместе с точками, бемолями и диезами определенную *частоту* и *длительность* звучания.

Все ноты музыкального произведения при помощи вертикальных отрезков разбиваются на группы. Совокупная длительность звучания нот каждой такой группки на протяжении всего конкретного произведения обычно строго постоянна и измеряется частями интервала (такта), или, как еще говорят, тактами разных размеров. Бывают такты: 4/4, 2/4, 6/8, 3/4, 3/2, 3/8, 1/4, 9/8 и др.

Между звучанием нот иногда необходимы паузы разных длительностей, их обозначения приведены в табл. 13.2.

Таблица 13.2. *Виды пауз*

Длительность, паузы, интервал	Изображение
1	
1/2	
1/4	
1/8	
1/16	
1/32	
1/64	

Использованием точек можно повышать длительность пауз аналогично повышению длительности звучания нот.

На рис. 13.3 в качестве примера показана нотная строка. На этом рисунке изображены группы нот, отделенные друг от друга вертикальными отрезками, — это и есть упомянутые ранее такты. Длительность такта обычно указывается только на первой строке произведения, сразу же за ключом.

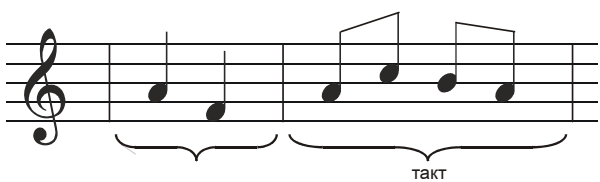


Рис. 13.3. Пример записи мелодии

В начале каждой строки могут ставиться также знаки альтерации. Ноты, которые находятся в той же строке, что и эти знаки, имеют, соответственно, повышенную или пониженную частоту (тон). Знаки альтерации, как уже отмечалось, часто расставляют еще и перед отдельными нотами. В этом случае глобальное действие знаков, расположенных в начале строки, на эти избранные ноты не распространяется.

Длительность звучания ноты устанавливается в диапазоне от 1 до 1/64 такта вторым аргументом процедуры `SOUND`. Пауза в звучании мелодии определяется процедурой `PLAY`, аргумент которой может иметь такие строковые значения: `"P1"`, `"P2"`, `"P3"`, ... , `"P64"`. Эти значения соответствуют задержкам в звучащей мелодии в интервале от 1 до 1/64 такта.

Заметим также, что важное значение в музыке имеет *темп* исполнения. Можно сказать, что он является характером мелодии. В табл. 13.3 приведены наиболее распространенные названия темпов (`Tn` — обозначение в BASIC).

Темп записывают в начале мелодии над первым тактом нотного стана. Иногда темп мелодии может меняться, и его записывают над нотным станом в месте, где нужно перейти на другой темп.

Процедура `PLAY` проигрывает музыкальные ноты. Строковый параметр является набором из одной или группы команд приведенного перечня.

Команды октавы и тона процедуры `PLAY` следующие:

- `O` (буква O) — задает номер текущей октавы, октавы могут иметь номера 0—6;
- `<` или `>` — перевод на одну октаву вниз или вверх;
- `A...G` — проигрывает заданную ноту текущей октавы;
- `N` — проигрывает заданный номер ноты (1—84) в диапазоне семи октав;
- `0` (ноль) — пауза.

Таблица 13.3. *Основные виды темпов*

Темп	`Tn`	Tempo
Очень медленно	73—80	Largo (протяжно), Grave (величаво)
Медлительно	78—82	Lento (несколько подвижнее, чем Largo)
Медленно, спокойно	83—86	Adagio (медленно)
Сдержанно, умеренно	87—93	Moderato (умеренно)
Не спеша	94—104	Andante (идущий, не спеша)
Неторопливо	105—137	Andantino (более скорое, чем Andante)
Подвижно	140—180	Animato (ускоряя движение, одушевленно)
В темпе вальса	171—186	Tempo di Valse
Живо	187—200	Vivace (подвижно)
Скоро, живо	190—210	Allegretto (довольно скоро), Allegro (весело, скоро)
Очень скоро	250—255	Presto (быстро), Prestissimo (очень быстро)

Примечание

All assai — весьма; темп увеличивается на 10 единиц.

`Tn` устанавливает темп звучания — число четвертных нот в 1 минуту. В `Tn` пределы `n` от 32 до 255, по умолчанию `n` = 120.

Команды длительности и типа могут быть такими:

- `ML` — вид исполнения *legato* (слитно, плавно);
- `MN` — вид исполнения *normal* (обычно);
- `MS` — вид исполнения *staccato* (отрывисто);

- P (Payza) — задает паузу между звучанием нот, параметр Payza может изменяться в интервале 1—64: P1 — пауза в целую ноту, P2 — пауза в 1/2 ноты и т. д;
- T (Temp) — задает темп исполнения — число исполняемых четвертей в минуту, параметр Temp может изменяться в диапазоне 32—255.

Команды режима следующие:

- MF — основное звучание;
- MB — фоновое звучание.

Команды изменения параметров ноты:

- # или + (диез);
- - (бемоль);
- . (длительность 3/2 от размера ноты).

Приведем комментарий к использованию этих команд. Прежде всего, необходимо отметить, что звучание ноты можно запрограммировать двумя способами. Первый из них состоит в указании номера октавы, например O4 (отмечаем, что команда O — это соответствующая буква, а не ноль), и названий нот в виде латинских букв: A (до), B (ре), C (ми), D (фа), E (соль), F (ля), G (си).

Переход к соседней октаве можно осуществить или путем непосредственного указания ее номера, как отмечалось ранее, или при помощи команд-символов > (увеличение номера октавы на единицу) или < (уменьшение номера октавы на единицу). Частоты нот каждой соседней октавы отличаются друг от друга в два раза. По умолчанию установлена октава O3, частоты нот которой имеют следующие значения в герцах:

- A (до) — 523,25;
- B (ре) — 587,33;
- C (ми) — 659,26;
- D (фа) — 698,46;
- E (соль) — 784,99;
- F (ля) — 880,00;
- G (си) — 987,77.

Во втором способе программирования нот частоты всех музыкальных звуков из всех семи октав пронумерованы в диапазоне от 1 до 84. Генерацию любого звука из этого дискретного спектра обеспечивает команда N(notu), в которой параметр N(notu) и есть номер выбранной ноты. При этом пауза между нотами обеспечивается командой N0 (отмечаем, что параметр 0 — это число ноль).

В табл. 13.4 приведен список команд, с помощью которых впоследствии запишем песню на BASIC.

Таблица 13.4. *Музыкальные команды оператора* PLAY

Команды октавы	
Октава	**Действие**
O*n*	Устанавливает текущую октаву *n*. Всего 7 октав, от 0 до 6
>	Повышает октаву на 1
<	Уменьшает октаву на 1

Таблица 13.4 *(продолжение)*

Команды тона	
Тон	**Действие**
A—G	Играет ноту A—G (до—си)
N*n*	Играет ноту *n* в пределах 0—84 (в семи возможных октавах всего 84 ноты); *n* = 0 означает паузу
Суффиксы	**Действие**
# или +	Следует за нотой и играет ее в режиме "диез"
-	Следует за нотой и играет ее в режиме "бемоль"
Команды длительности	
Длительность	**Действие**
L*n*	Устанавливает длительность каждой ноты. L4 — четвертная нота, L1 — целая нота и т. д. Пределы *n* — от 1 до 64. Длительность может следовать за нотой, если не нужно задавать длительность всей последовательности. Например, A16 эквивалентно L16A
MN	Устанавливает "нормальный стиль", т. е. ноты играются 7/8 времени, определенного длительностью L
ML	Устанавливает "стиль легато", т. е. ноты играются полный период, установленный командой L
MS	Устанавливает "стиль стаккато", т. е. ноты играются 3/4 периода, указанного командой L
Команды темпа	
Темп	**Действие**
P*n*	Устанавливает паузу длительностью *n*, в пределах 1—64. Параметр *n* должен быть задан явно
T*n*	Устанавливает темп звучания — число четвертных нот в одной минуте. Пределы *n* — 32—255. По умолчанию *n* = 120. Однако некоторые ноты могут не звучать при высоком темпе (например, L16 при T255)
. (точка)	Точка после ноты указывает на то, что она должна звучать 3/2 времени, указанного L. После ноты может стоять несколько точек, каждая из которых добавляет половину предыдущей длительности (например, A. Звучит 1 + 1/2 или 3/2 длительности; A.. звучит 1 + 1/2 + 1/4 или 7/4. Точки могут стоять и после паузы P, регулируя ее длительность
Переключение звучания на основное или фоновое	
Операция	**Действие**
MF	Устанавливает музыку, генерируемую операторами PLAY и SOUND, в основное звучание, когда последующая нота звучит только после завершения предыдущей, а действие других операторов приостанавливается до завершения звучания
MB	Музыкальные операторы генерируют звук в фоновом режиме. Это значит, что каждая нота или звук помещаются в буфер, позволяя другим операторам выполняться, пока звучит музыка. Максимальное количество нот, помещающихся в буфере, равно 32

Таблица 13.4 *(окончание)*

Вызов подкоманды	
"X" + VARPTR$(*строка*)	Выполняет подкоманду, указанную в строке

Далее, нам нужно перевести ноты в программу BASIC. Для этого используем приведенные ранее команды.

Пример 13.1. Мелодия к русскому романсу "Динь-динь-динь"

В качестве примера составим программу на языке BASIC, воспроизводящую мелодию старинного русского романса на слова и музыку Е. Юрьева "Динь-динь-динь":

1. В лунном сиянье снег серебрится
   Вдоль по дороге троечка мчится
   "Динь-динь-динь, динь-динь-динь"
   Колокольчик звенит.
   Этот звон, этот звук
   Много мне говорит.
2. В лунном сиянье ранней весною
   Вспомнились встречи, друг мой, с тобою...
   Колокольчиком твой голос юный звенел...
   "Динь-динь-динь, динь-динь-динь" —
   О любви сладко пел...
3. Вспомнился зал мне с шумной толпою,
   Личико милой с белой фатою...
   "Динь-динь-динь, динь-динь-динь" —
   Звон бокалов звучит...
   С молодою женой мой соперник стоит!

Нотная запись песни приведена на рис. 13.4.

Изучив список команд, можно приступать к работе. Для того чтобы перевести ноты в программу BASIC, следует знать их длительность, октаву, темп и т. д. Для тех, кто не имеет начального музыкального образования, целесообразно делать промежуточный вариант.

Промежуточной операцией для перевода нот в команды BASIC является запись нот на листе бумаги в соответствии с их длительностью, октавой.

Чтобы распознать ноту, следует воспользоваться рис. 13.2, с помощью которого определяется название ноты и октава. Далее по табл. 13.1 можно узнать длительность ноты. Длительность пауз следует из табл. 13.2.

Для рассматриваемой песни имеем следующий промежуточный вариант:

```
СИ (ЦЕЛАЯ С ТОЧКОЙ)
СОЛЬ (ЦЕЛАЯ С ТОЧКОЙ)
ФА 4
СИ (ЦЕЛАЯ С ТОЧКОЙ)
СОЛЬ (ЦЕЛАЯ С ТОЧКОЙ)
```

```
СИ (ЦЕЛАЯ С ТОЧКОЙ)
ЛЯ (ЦЕЛАЯ С ТОЧКОЙ)
ЛЯ 4 СИ 4 ДО 4 (2 ОКТАВА)
СИ (ЦЕЛАЯ)
СОЛЬ 4
МИ (ЦЕЛАЯ С ТОЧКОЙ)
ДО (2 ОКТАВА ЦЕЛАЯ С ТОЧКОЙ)
ДО 4 (2 ОКТАВА) СИ 4 ЛЯ 4
СИ (ЦЕЛАЯ С ТОЧКОЙ)
СОЛЬ (ЦЕЛАЯ С ТОЧКОЙ)
ФА (ЦЕЛАЯ С ТОЧКОЙ)
ФА 4 СОЛЬ 4 РЕ 4 (ДИЕЗ)
ФА (ЦЕЛАЯ С ТОЧКОЙ)
МИ 4 СИ 4 СОЛЬ 4
МИ (ЦЕЛАЯ С ТОЧКОЙ)
ПАУЗА 4 РЕ 4 (ДИЕЗ) МИ 4
ФА (ЦЕЛАЯ С ТОЧКОЙ)
ПАУЗА 4 ФА 4 СОЛЬ 4
ЛЯ (ЦЕЛАЯ С ТОЧКОЙ)
ЛЯ 4 СИ 4 ДО 4 (2 ОКТАВА)
СИ (ЦЕЛАЯ С ТОЧКОЙ)
ПАУЗА 4 СИ 4 СОЛЬ 4
МИ (ЦЕЛАЯ С ТОЧКОЙ)
ПАУЗА 4 ДО 4 (2 ОКТАВА С ТОЧКОЙ) СИ 8
ЛЯ (ЦЕЛАЯ С ТОЧКОЙ)
ПАУЗА 4 ДО 4 (2 ОКТАВА) ЛЯ 4
СИ (ЦЕЛАЯ С ТОЧКОЙ)
ПАУЗА 4 СОЛЬ 4 ФА 4
МИ (ЦЕЛАЯ С ТОЧКОЙ)
```

Теперь можно начинать создание программы (программа 13.1). Для ввода текста в строках 20—170 используем оператор `PRINT`. Так как в песне 3 куплета, то для того, чтобы повторить мелодию 3 раза, нужно организовать цикл `FOR...NEXT` в строках 180—250. В теле данного цикла используем оператор `PLAY` для ввода самого романса.

Программа 13.1

```
10 CLS
20 PRINT "В лунном сиянье снег серебрится"
30 PRINT "Вдоль по дороге троечка мчится"
40 PRINT "Динь-динь-динь, динь-динь-динь"
50 PRINT "Колокольчик звенит"
60 PRINT "Этот звон, этот звук"
70 PRINT "Много мне говорит"
80 PRINT "В лунном сиянье ранней весною"
90 PRINT " Вспомнились встречи друг мой с тобою"
100 PRINT "Колокольчиком твой голос юный звенел..."
110 PRINT "Динь-динь-динь, динь-динь-динь -"
120 PRINT "О любви сладко пел..."
130 PRINT "Вспомнился зал мне с шумной толпою,"
140 PRINT "Личико милой с белой фатою"
150 PRINT "Динь-динь-динь, динь-динь-динь"
160 PRINT "Звон бокалов звучит"
```

```
170 PRINT "С молодою женой мой соперник стоит!"
180 FOR i = 1 TO 3
190 PLAY "T220 g1 e1.  d4 g1. e1. g1. f1."
200 PLAY " f4 g4 >a4< g1 e4 c1.> a1.a4<g4 f4"
210 PLAY " g1. e1. d1. d4 e4 b4 d1."
220 PLAY "c4 g4 e4 c1. p4 b4 c4 d1. p4 d4 e4"
230 PLAY "f1.f4 g4>a4<g1.p4 g4 e4 c1."
240 PLAY "p4>a4<g8 f1.p4>a4< f4 g1.p4 e4 d4 c1."
250 NEXT i
260 END
```

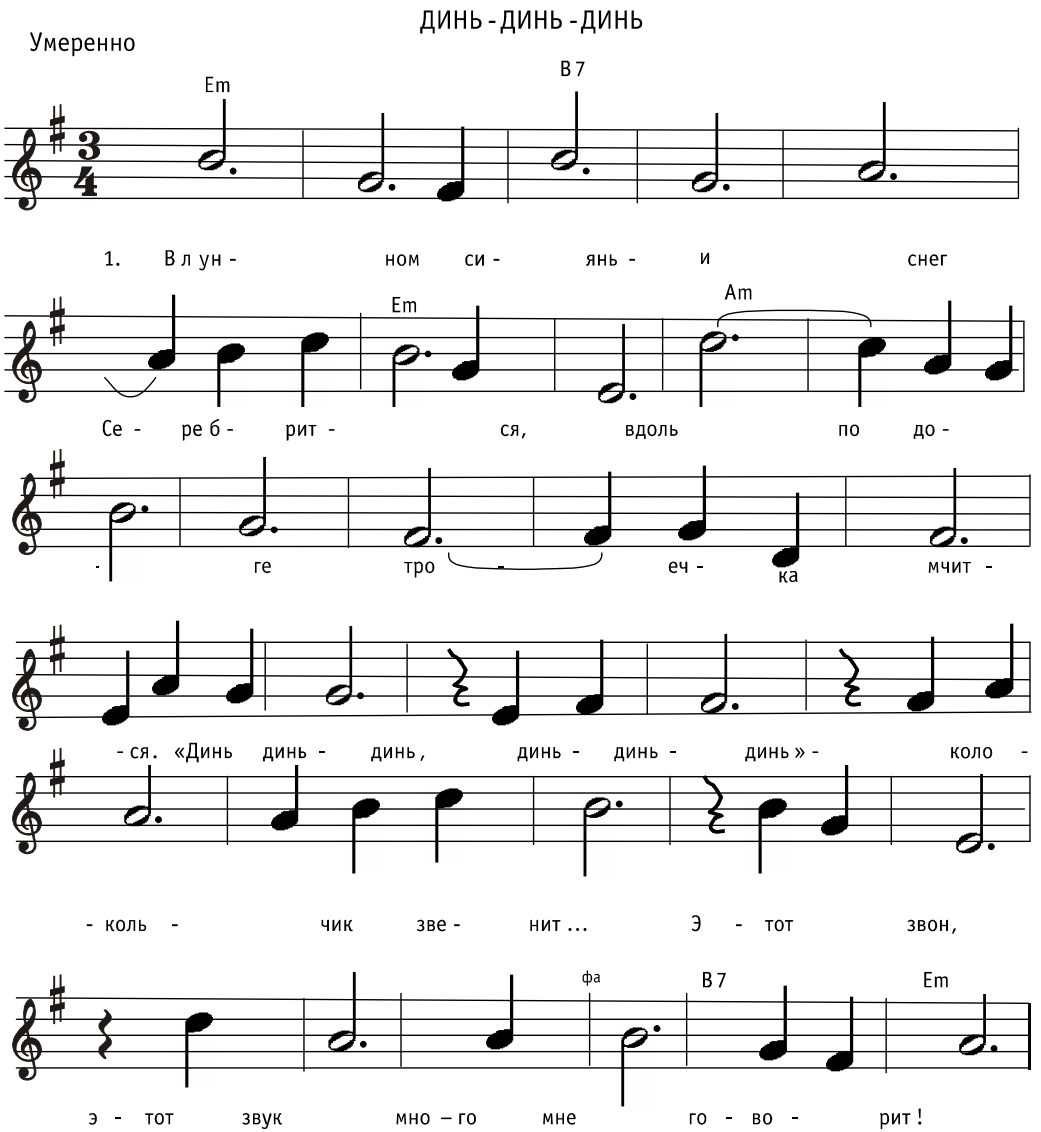


Рис. 13.4. Нотная запись русского романса "Динь-динь-динь"

Записать в BASIC мелодию можно и с помощью другого оператора — SOUND.

SOUND — оператор ввода/вывода, генерирующий звук через динамик:

```
SOUND n, d
```

где n — частота звука в циклах/сек или в герцах, числовое выражение в пределах 37—32 767;

d — длительность, числовое выражение в пределах 0—65 535, число отсчетов системного таймера, определяющего продолжительность звучания. В одной секунде 18.2 отсчета таймера.

Если длительность равна нулю, то звук, генерируемый текущим оператором SOUND, выключается.

В табл. 13.5 приведены соответствующие нотам частоты.

Таблица 13.5. *Соответствие звуковых частот нотам ряда октав*

Октава	Большая	Малая	Первая	Вторая
Нота	**Частота**			
До	131	262	523	1047
Ре	147	294	587	1174
Ми	165	330	659	1319
Фа	175	349	698	1396
Соль	196	392	785	1568
Ля	220	440	880	1760
Си	247	494	988	1975

Поскольку диез повышает звучание ноты на полтона, то частота будет равняться частоте на середине соответствующего диапазона, например, для ми-диез — середине диапазона между ми и фа. Аналогично, бемоль понижает на полтона, значит, частота ми-бемоль будет на середине диапазона между ми и ре. В частности, для малой октавы имеем:

- до-диез соответствует частота 277 Гц;
- ре-диез — частота 311 Гц;
- фа-диез — частота 370 Гц;
- соль-диез — частота 415 Гц;
- ля-диез — частота 466 Гц и т. д.

Пример 13.2. Мелодия к песне о бедном зайчике

В книге [8] приведен пример записи мелодии с использованием оператора SOUND песенки о незабвенном зайчике, опрометчиво вышедшем погулять и нарвавшемся на нехорошего дядю (программа 13.2).

Программа 13.2

```
'Программа Zajka
CONST Takt = 18.2, Takt2 = 9.1, Takt4 = 4.55, Takt8 = 2.275
```

```
'Частоты измеряются в герцах
CONST nDo = 131, vDo = 262
CONST nDoD = 139, vDoD = 277
CONST nRe = 147, vRe = 294
CONST nReD = 156, vReD = 311
CONST nMi = 165, vMi = 330
CONST nFa = 175, vFa = 349
CONST nFaD = 185, vFaD = 370
CONST nSol = 196, vSol = 392
CONST nSolD = 208, vSolD = 415
CONST nLa = 220, vLa = 440
CONST nLaD = 233, vLaD = 466
CONST nSi = 247, vSi = 494
CONST vvDo = 523
SOUND nDo, Takt8: SOUND nDo, Takt8
SOUND nRe, Takt8: SOUND nRe, Takt8:
SOUND nMi, Takt8: SOUND nMi, Takt8
SOUND nFa, Takt4: SOUND nMi, Takt8: SOUND nFa, Takt8
SOUND nSol, Takt8: SOUND nFa, Takt8:
SOUND nMi, Takt8: SOUND nRe, Takt8
SOUND nRe, Takt4: SOUND nDo, Takt8: SOUND nDo, Takt8
SOUND nRe, Takt8: SOUND nRe, Takt8:
SOUND nMi, Takt8: SOUND nMi, Takt8
SOUND nFa, Takt8: SOUND nFa, Takt8:
SOUND nSol, Takt8: SOUND nSol, Takt8
SOUND nLa, Takt8: SOUND nLa, Takt8:
SOUND nSi, Takt8: SOUND nSi, Takt8
SOUND vDo, Takt8: SOUND vDo, Takt8: SOUND vDo, Takt8: PLAY "P8"
SOUND nSi, Takt8: PLAY "P8": SOUND nLa, Takt8: SOUND nLa, Takt8
SOUND nSol, Takt4: SOUND nFa, Takt8: SOUND nFa, Takt8
SOUND nMi, Takt8: SOUND nMi, Takt8:
SOUND nRe, Takt8: SOUND nRe, Takt8
SOUND nDo, Takt4
END
```

Причем, автор указанной книги [8] предлагает читателю модернизировать программу, добавив слова, что и было сделано в программе 13.3.

Программа 13.3

```
'Zayka.bas
10 CLS
20 PRINT : PRINT TAB(32); "Зайчик": PRINT
30 PRINT TAB(22); "Раз, два, три, четыре, пять."
40 PRINT TAB(22); "Вышел зайчик погулять."
50 PRINT TAB(22); "Вдруг охотник выбегает,"
60 PRINT TAB(22); "Прямо в зайчика стреляет -"
70 PRINT TAB(22); "Пиф-паф! — Не попал,"
80 PRINT TAB(22); "Быстрый зайчик убежал."
90 FOR k = 1 TO 42
```

```
100 IF k = 7 OR k = 14 OR k = 35 OR k = 42 THEN t = 4.55 ELSE t = 2.275
110 DATA 131,131,147,147,165,165,175,165,175,196,175,165,147,147
120 DATA 131,131,147,147,165,165,175,175,196,196,220,220,247,247
130 DATA 262,262,262,247,220,220,196,175,175,165,165,147,147,131
140 READ n
150 SOUND n, t
160 IF k = 31 OR k = 32 THEN SOUND 32000, t
170 NEXT k
180 END
```

Отметим, что модернизация была проделана чисто формально. Как кажется авторам, в примере из книги [8] напутано с октавами. Как видно из табл. 13.5, в программе 13.2 использованы частоты, соответствующие нотам Большой октавы. А в нотной записи мелодии приведены ноты первой октавы. Оставим это на совести автора книги [8].

Пример 13.3. Мелодия песни Ю. Визбора "Ты у меня одна"

Рассмотрим пример разработки программы, воспроизводящей мелодию песни, приведенной на рис. 13.5.

Используем в этой программе оператор SOUND. Осуществим первоначально, как было проделано выше, запись промежуточного варианта с использованием рис. 13.2 и табл. 13.1 и 13.2:

- 1 октава Ре8 Ре8 Ре8 Соль4 Си4-бемоль Соль4 Пауза 8 Пауза 4 с точкой;
- 1 октава Ре8 Ре8 Ре8 Малая октава Си4-диез 1-я октава Ре8 До4 Пауза 8 Пауза 4 с точкой;
- Малая окт. Ля8 Ля8 Ля8 1 окт. До8 Пауза 8 До8 Ми8-бемоль Пауза 8 Пауза 4 с точкой;
- 1 октава До8-диез До8 До8 Соль8 Пауза 8 Фа8-диез Ля4 Пауза 8 Пауза 4 с точкой;
- 1 октава Ре8 Ре8 Пауза 16 Ре16 Соль8 Пауза 8 Си8-бемоль Соль4 Пауза 8 Пауза 4 с точкой;
- 1 октава Ре8 Пауза 8 Ре16 Ре16 Малая октава Си8 Пауза 8 с точкой 1-я октава Ре16 Ре8 До4 Пауза 4 с точкой;
- Малая октава Ля8 Пауза 8 Ля16 Ля16 1-я октава До4 До8 Ми4-бемоль Пауза 8 Пауза 4 с точкой;
- 1 октава Ре8 Ре8 Ре8 Фа4-диез Ре8 Соль4 с точкой Пауза 4 с точкой.

Итак, по фрагментам число музыкальных элементов, требующих оператора SOUND, будет:

1. 8;
2. 8 (16);
3. 9 (25);
4. 9 (34);
5. 10 (44);
6. 10 (54);
7. 9 (63);
8. 7 (70).

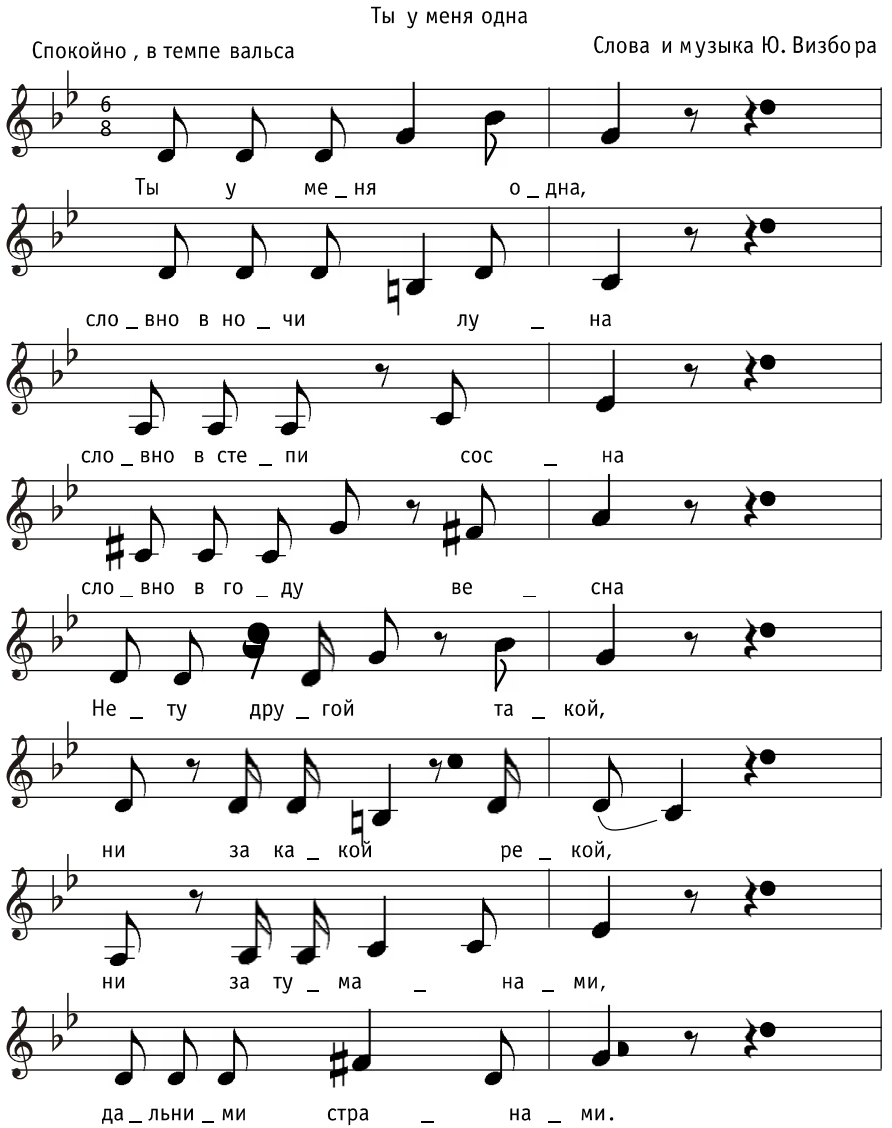


Рис. 13.5. Мелодия песни "Ты у меня одна"

Следовательно, для записи мелодии в BASIC с помощью оператора `SOUND` требуется цикл с числом шагов `k` = 70. По длительности имеем 8 вариантов:

1. Четвертная (`CASE 4, 6, 12, 14, 23, 32, 42, 53, 59, 61, 67`).
2. Четвертная с точкой (`CASE 69`).
3. Шестнадцатая (`CASE 38, 47, 48, 51, 57, 58`).
4. Восьмерная (`CASE ELSE`).
5. Четверная пауза с точкой (`CASE 8, 16, 25, 34, 44, 54, 63, 70`).
6. Восьмерная пауза (`CASE 7, 15, 21, 24, 30, 33, 40, 43, 46, 56, 62`).

7. Восьмерная пауза с точкой (CASE 50).
8. Шестнадцатая пауза (CASE 37).

Очевидно, что можно объединить варианты 4 и 6, вариант 2 с вариантом 5, а также варианты 3 и 8. Для реализации ввода длительности лучше всего использовать условный оператор SELECT CASE. Поскольку восьмерных нот в мелодии больше всего, то лучше их проводить через CASE ELSE. Ввод частот нот осуществим, как и в программе 13.3, посредством пары операторов READ/DATA, данные для которых запишем, используя табл. 13.5.

Поскольку песня содержит три куплета, то используется двойной цикл FOR...NEXT. Причем внутренний цикл реализует звучание куплета, а внешний — его повторение требуемое число раз.

Программа 13.4

```
'13Pecna.bas
10 CLS
20 PRINT : PRINT TAB(22); "Ты у меня одна": PRINT
30 PRINT "  1 Ты у меня одна,           2 В инее провода,"
40 PRINT "    Словно в ночи луна,        В сумерках города."
50 PRINT "    Словно в году весна,       Вот и взошла звезда,"
60 PRINT "    Словно в степи сосна.     Чтобы светить всегда,"
70 PRINT "    Нету другой такой           Чтобы гореть в метель,"
80 PRINT "    Ни за какой рекой,           Чтобы стелить постель,"
90 PRINT "    Ни за туманами,               Чтобы качать всю ночь"
100 PRINT "    Дальними странами.        У колыбели дочь.": PRINT
110 PRINT "                  3 Вот поворот какой"
120 PRINT "                     Делается рекой."
130 PRINT "                     Можешь отнять покой,"
140 PRINT "                     Можешь махнуть рукой,"
150 PRINT "                     Можешь отдать долги,"
160 PRINT "                     Можешь любить других,"
170 PRINT "                     Можешь совсем уйти,"
180 PRINT "                     Только свети, свети!"
190 t0 = 18.2 * 1.8
200 FOR i = 1 TO 3
210 RESTORE
220 FOR k = 1 TO 70
230 SELECT CASE k
       CASE 4, 6, 12, 14, 23, 32, 42, 53, 59, 61, 67: t = t0 / 4
       CASE 8, 16, 25, 34, 44, 54, 63, 69, 70: t = (t0 / 4) * 3 / 2
       CASE 37, 38, 47, 48, 51, 57, 58: t = t0 / 16
       CASE 50: t = (t0 / 8) * 3 / 2
       CASE ELSE: t = t0 / 8
240 END SELECT
250 DATA 587,587,587,785,934,785,0,0,   587,587,587,508,587,523,0,0
260 DATA 440,440,440,523,0,523,623,0,0,  555,523,523,785,0,741,880,0,0
270 DATA 587,587,0,587,785,0,934,785,0,0,  587,0,587,587,494,0,587,587,523,0
280 DATA 440,0,440,440,523,523,623,0,0,  587,587,587,741,587,785,0
290 READ n
300 SOUND n, t
310 NEXT k, i
320 END
```

Если в процедуре PLAY имеется специальная команда T для установки необходимого темпа, то при использовании оператора SOUND темп приходится задавать путем введения корректирующего множителя (строка 190 программы 13.4, где множитель 1.8).

Оператор SOUND можно использовать не только для записи мелодий, но и для создания звуковых эффектов. В книге [3] имеется программа, содержащая 17 вариантов подобных эффектов. Приводим ее с небольшими доработками.

Программа 13.5

```
'13ZvEff.bas     Звуковые эффекты
10 DEFINT A-Z
20 CLS
30 PRINT : PRINT TAB(10); "Звуковые эффекты": PRINT
40 PRINT " 1. Oh"; TAB(15); " 7. Waver"; TAB(28); "13. Vrowr"
50 PRINT " 2. Space"; TAB(15); " 8. Who"; TAB(28); "14. Zhou"
60 PRINT " 3. Gurgle"; TAB(15); " 9. Mew"; TAB(28); "15. Art"
70 PRINT " 4. Spectre"; TAB(15); "10. Phone"; TAB(28); "16. Coo"
80 PRINT " 5. Grup"; TAB(15); "11. Siren"; TAB(28); "17. Squawk"
90 PRINT " 6. Chirp"; TAB(15); "12. Zhoup"; TAB(28); "18. Все"
100 INPUT "   Выберите номер эффекта (0 — выход) — ", v: vv = v
110 IF v < 0 OR v > 18 THEN  ELSE 130
120 PRINT : PRINT "   Такого варианта нет": SLEEP 2: GOTO 20
130 e = 0
140 DO
   '----- Выбор 18. Все ------------------------------------------
150 IF v = 18 THEN
160 IF e = 0 THEN vv = 0: e = 1
170 vv = vv + 1
180 SLEEP 1
190 LOCATE 10, 44: PRINT "("; vv; ")"
200 END IF
   '-------------------------------------------------------------
210 SELECT CASE vv
CASE 0: EXIT DO                                 'Выход
CASE 1: FOR i = 800 TO 2000 STEP 100            'Oh
        SOUND i, .2
        NEXT i
        FOR i = 2000 TO 50 STEP -100
        SOUND i, .2
        NEXT i
CASE 2: FOR i = 1000 TO 40 STEP -20             'Space
        SOUND i, .2
        NEXT i
CASE 3: FOR i = 10 TO 50 STEP 10                'Gurgle
        FOR j = 50 TO 10 STEP -10
        SOUND i ^ 2 + j ^ 2, .1
        NEXT j, i
CASE 4: FOR z = 1 TO 100                        'Spectre
        SOUND (SIN(z) + 40) * 50, .2
        NEXT z
CASE 5: FOR i = 10 TO 50 STEP 10                'Grup
        FOR j = 50 TO 10 STEP -10
```

```
        SOUND i * j, .1
        NEXT j, i
CASE 6: FOR i = 30 TO 60 STEP 10                'Chirp
        FOR j = 60 TO 30 STEP -10
        SOUND i ^ 2 + j ^ 2, .2
        NEXT j, i
CASE 7: FOR z = 1 TO 45                         'Waver
        SOUND (SIN(z) + 20) * 30, .2
        NEXT z
CASE 8: FOR z = 100 TO 80 STEP -1               'Who
        SOUND (TAN(z) + 36) * 25, .8
        SOUND (SIN(z) + 20) * 50, .4
        NEXT z
CASE 9: FOR z = 100 TO 80 STEP -1               'Mew
        SOUND (TAN(z) + 50) * 25, .4
        NEXT z
CASE 10: FOR z = 1 TO 10                        'Phone
         SOUND 1195, .4
         SOUND 2571, .4
         NEXT z
CASE 11: FOR z = 1 TO 3                         'Siren
         SOUND 550, 9
         SOUND 400, 9
         NEXT z
CASE 12: FOR z = 3 TO 12                        'Zhoup
         SOUND 120 + z ^ 4, .1
         SOUND 0, .1
         NEXT z
CASE 13: FOR z = 12 TO 3 STEP -1                'Vrowr
         SOUND 120 + z ^ 4, .1
         SOUND 0, .1
         NEXT z
CASE 14: FOR i = 40 TO 15 STEP -1               'Zhou
         FOR j = 90 TO 50 STEP -10
         SOUND i * j, .1
         NEXT j, i
CASE 15: FOR z = 1 TO 10                        'Art
         SOUND 1195 — 50 * z, .3
         SOUND 1195 + 50 * z, .3
         NEXT z
CASE 16: FOR z = 0 TO 150 STEP 10               'Coo
         SOUND 1295 — z, .4
         SOUND 1095 + z, .4
         NEXT z
CASE 17: FOR i = 1 TO 20                        'Squawk
         FOR j = 50 TO 150 STEP 50
         SOUND i * j, .1
         NEXT j, i
220 END SELECT
230 IF v < 18 THEN CLS : GOTO 20
240 IF v = 18 AND vv = 17 THEN CLS : GOTO 20    'Случай 18. Все
250 LOOP
260 END
```

ПРИЛОЖЕНИЕ 1

Язык программирования BASIC

Название языка программирования BASIC — это первые буквы английских слов Beginner's All-purpose Symbolic Instruction Code (Многоцелевой язык программирования для начинающих). Впрочем, имеется и другой перевод этого названия — на английском языке слово *basic* означает "основной, базовый".

Созданный в начале 60-х годов XX века двумя профессорами Дартмунского университета Джоном Кенеми и Томасом Куртцом, BASIC превратился в современный язык высокого уровня. Он продолжает оставаться простым и доступным для всех пользователей, как для любителей, так и для профессионалов. По словам президента фирмы Microsoft Билла Гейтса, BASIC переживет все другие языки программирования. И жизнь подтверждает это утверждение. На сегодняшний день BASIC — самый распространенный язык программирования для IBM PC. По данным журнала "PC Word" свыше 70% программистов владеют тем или иным диалектом BASIC.

Язык программирования BASIC постоянно совершенствуется, снабжается всем необходимым для профессиональной разработки программ: поддержка модульного и структурного программирования, эффективная среда разработки, создание исполняемых модулей, мощные средства отладки, встроенная справка, дополнительные библиотеки процедур различного назначения — от пользовательского интерфейса до управления базами данных.

При этом BASIC сохраняет свою привлекательность для начинающих и непрофессиональных программистов, прежде всего, за счет более простой технологии работы в среде интерпретатора и наличия большого числа операторов высокого уровня (графика, обработка строковых переменных, работа с аппаратурой и пр.). BASIC — признанный лидер по скорости разработки и отладки программ. Это качество особенно требуется в тех многочисленных областях прикладных разработок, в которых знание предметной области является зачастую более важным, чем высокая квалификация программиста.

Фирма Microsoft является не единственным разработчиком систем BASIC. Существуют также версии GFA, True, Power, Z и некоторые другие, но они не получили столь широкое распространение — отношение объема продажи языков BASIC фирмы Microsoft к реализациям BASIC остальных фирм составляет 12:1 (рис. П1.1).

Кратко остановимся на трех поколениях языка BASIC фирмы Microsoft:

1. Поколение интерпретаторов (GWBASIC, BASICA).
2. Поколение QuickBASIC.
3. Поколение Visual BASIC.

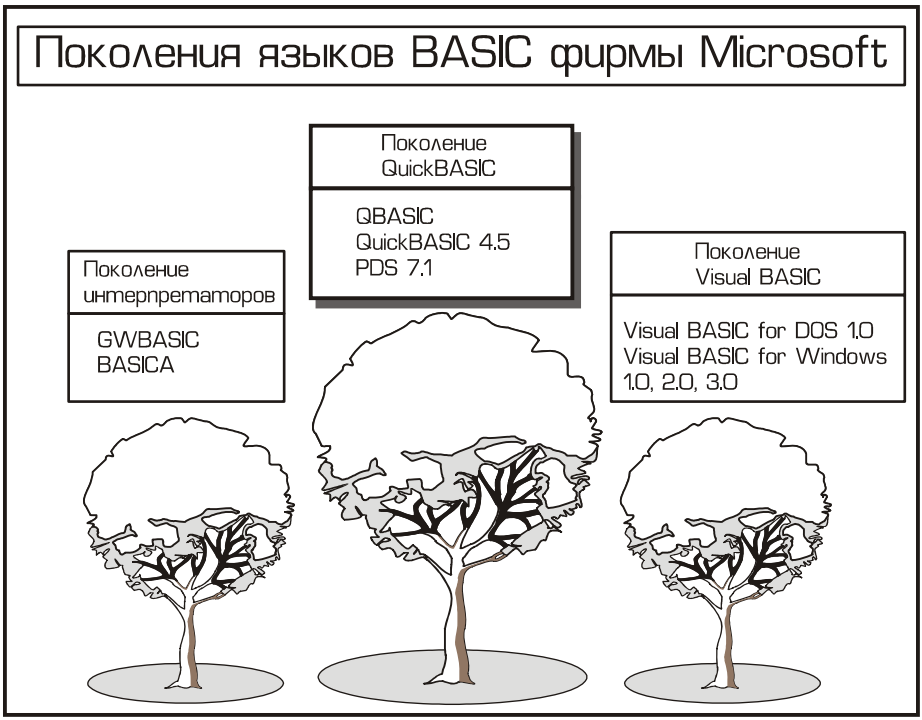


Рис. П1.1. Поколения языков BASIC

GWBASIC — первое поколение языка

Представления о версиях BASIC фирмы Microsoft большинства пользователей персональных компьютеров 20 лет назад, как правило, были ограничены системой GWBASIC, которая входила в состав MS-DOS до версии 4.01 включительно. Сегодня GWBASIC является морально устаревшей системой для персональных компьютеров, потому что:

- отсутствует современная среда разработки. Редактирование текста с помощью нумерации строк было предложено, когда в качестве диалогового устройства использовалась пишущая машинка;
- работа только в среде интерпретатора предполагает при каждом запуске программы каждую строку программы переводить в машинные коды. Это очень замедляет выполнение готовой программы;
- используется только 64 Кбайт оперативной памяти, отсутствуют модули процедур и управляющих структур.

QuickBASIC — второе поколение языка

Создание Microsoft QuickBASIC (сокращенное обозначение — QBASIC) в середине 80-х годов прошлого века произвело настоящую революцию в мире BASIC, что позволило языку занять достойное место среди средств разработки серьезных прикладных систем. На сегодняшний день QBASIC стал стандартом de facto для языка BASIC.

Первые версии QBASIC появились в 1985 году, а последняя, 4.5 — была создана в 1988 году. С точки зрения функциональных возможностей, QBASIC 4.5 практически идентичен предыдущей версии 4.0 (1987 год), но имеет более удобную среду программирования. У него имеется мощная справочная система, а также возможность управления опциями среды (цвет, имена каталогов и пр.). Внешне непохожий на традиционный BASIC, QBASIC в очень высокой степени обеспечивает совместимость на уровне исходного кода с предыдущими версиями (GWBASIC, BASIC, Turbo).

В QBASIC в достаточно полной мере реализованы идеи структурного и модульного программирования, возможности использования процедур и функций. Наряду с созданием механизма применения двоичных библиотек, а следовательно, и готовых программных модулей, в том числе написанных на других языках, это обеспечило создание достаточно больших программных систем.

Простая и компактная по сравнению с современными пакетами-монстрами Quick-система может быть рекомендована в качестве первого этапа работы для всех, кто начинает осваивать работу с семейством Microsoft BASIC. Начиная с версии MS-DOS 5.0, вместо устаревшего GWBASIC фирма Microsoft стала поставлять систему QBASIC, которая представляет собой усеченный вариант QuickBASIC без компилятора и некоторых возможностей модульного программирования. Ее русифицированный вариант входит в состав соответствующей локализованной версии MS-DOS 5.0 и 6.0.

В 1989 году появилась Microsoft Basic Profession Development System (система для профессиональной разработки) версии 7.0, а на следующий год ее сменила версия 7.1. Basic PDS, в отличие от QBASIC 4.5, позволяет создавать более мощные программные комплексы и расширяет круг решаемых задач за счет использования дополнительных возможностей процессора и оперативной памяти, новых средств разработки программ, встроенной системы управления большими базами данных, а также повышения эффективности программного кода (объем памяти, быстродействие). Кроме новых возможностей, в PDS исправлен ряд ошибок QBASIC, в частности, нет проблем с вводом прописной русской буквы "р". Несмотря на появление PDS и Visual-систем, фирма Microsoft продолжает коммерческую реализацию QBASIC 4.5 (мировая цена — $99).

Visual BASIC — третье поколение языка

Первая версия новой системы Visual BASIC (VB) появилась в 1991 году под лозунгом "теперь и начинающие программисты могут свободно работать в Windows". Главной особенностью системы является принципиально новая логика и возможность использования средств системы Windows в форме обращения к ее собственным функциям.

Уже через год после появления версии VB/Win 1.0 была выпущена версия 2.0, а летом 1993 года — 3.0. В 1992 году была также подготовлена система VB for DOS версии 1.0. В настоящее время фирма Microsoft поддерживает и развивает Visual BASIC как одну из своих основных систем программирования, рассчитанную для массового и профессионального использования.

Появление VB связано с популяризацией идей событийно-управляемого и визуального программирования в среде Windows. Язык VB является развитием QuickBASIC. Несмотря на появление новых операторов, в принципе это тот же вариант языка программирования, но с некоторыми принципиальными ограничениями.

ПРИЛОЖЕНИЕ 2

Сообщения об ошибке

Таблица П2.1. *Сообщения об ошибке и их описание*

Код	Сообщение об ошибке и возможная причина
1	NEXT без FOR (NEXT without FOR) Для окончания цикла `NEXT` нет соответствующего заголовка `FOR`. Количество `FOR` и `NEXT` должны совпадать
2	Синтаксическая ошибка (Syntax error) Оператор содержит грамматическую ошибку в написании ключевого слова или ошибку в пунктуации, либо есть непарные скобки или другие нарушения правил синтаксиса языка BASIC
3	RETURN без GOSUB (RETURN without GOSUB) Для оператора возврата из подпрограммы `RETURN` нет соответствующего обращения к подпрограмме `GOSUB`
4	Нет данных (Out of DATA) В операторе `DATA` нет данных. Посчитайте количество данных в операторе `DATA` и количество считываний из него оператором `READ`. Посмотрите внимательно, не поставили ли вы при перечислении данных в каком-нибудь месте точку вместо запятой
5	Неверный вызов функции (Illegal function call) Возникает чаще всего при попытке извлечения квадратного корня из отрицательного числа или появления отрицательного числа или 0 под знаком логарифма, а также применения графических операторов без включения графического режима `SCREEN`. Вообще же подобное сообщение возникает при попытке вызова функции с недопустимым параметром
6	Переполнение (Overflow) Числовая переменная или строковая константа выходят за пределы допустимого диапазона (например, в знаменателе получается очень малая величина или при работе с возведением в степень). Проверьте и измените значение при необходимости
7	Не хватает памяти (Out of memory)
8	Метка не определена (Label not defined) Для операторов `GOTO` или `GOSUB` задается переход на несуществующую метку
9	Индекс вне диапазона (Subscript out of range) Сообщение возникает при работе с массивами, когда индекс какого-либо элемента массива превышает его объявленный в операторе `DIM` размер, а также в том случае, когда массив занимает в памяти объем более 64 Кбайт. Эта ошибка появляется также, если в формуле, оперирующей с элементами массива, они заменены другими переменными (`x(i)` заменен на простой `x`)

Таблица П2.1 *(продолжение)*

Код	Сообщение об ошибке и возможная причина
10	Повторяющееся определение (Duplicate definition) Может возникнуть, если элемент массива, объявленного в операторе `DIM`, фигурирует далее (в формуле или выражении) в несвязанном или неправильно связанном виде
11	Деление на ноль (Division of zero) Выражение в знаменателе после подстановки значений переменных и вычислений, видимо, обращается в ноль
12	Ошибка в режиме управления (Illegal in direct mode)
13	Несоответствие типа (Type mismatch)
14	В строке нет места (Out of string space)
16	Слишком сложная строковая формула (String formula too complex)
17	Невозможно продолжить (Cannot continue)
18	Функция не определена (Function not defined) Возможно, используемая функция не определена оператором `DEF FN`, или допущена ошибка при определении или вызове функции
19	Нет RESUME (No RESUME)
20	RESUME без ошибки (RESUME without error)
24	Устройство в тайм-ауте (Device timeout)
25	Ошибка устройства (Device fault)
26	FOR без NEXT (FOR without NEXT) Для заголовка цикла `FOR` нет соответствующего окончания цикла `NEXT`. Количество `FOR` и `NEXT` должно совпадать
27	Нет бумаги (Out of paper)
29	WHILE без WEND (WHILE without WHILE) Для ключевого слова `WHILE` нет соответствующего слова `WEND`
30	WEND без WHILE (WEND without WHILE) Для ключевого слова `WEND` нет соответствующего слова `WHILE`
33	Повторяющаяся метка (Duplicate label) При расстановке меток допущен повтор одной и той же метки в разных местах программы. Обычно возникает при редактировании текста программы копированием
35	Подпрограмма не определена (Subprogram not defined) Сообщение возникает при попытке обращения к несуществующей подпрограмме
37	Ошибка счетчика аргументов (Argument-count mismatch)
38	Массив не определен (Array not defined) Попытка работать с элементами массива, который не был объявлен оператором `DIM`
39	Требуется CASE ELSE (CASE ELSE expected)
40	Необходима переменная (Variable required) Ошибка возникает при попытке записи иных элементов программы в том месте, где должна быть переменная (попытка заменить `x` русской буквой `x`)
50	Переполнение FIELD (FIELD overflow)

Таблица П2.1 *(продолжение)*

Код	Сообщение об ошибке и возможная причина
51	Внутренняя ошибка (Internal error) Чаще всего это неверная работа компьютера, реже — смысловые ошибки программы, не сразу различимые на первый взгляд. Часто требуется ручная прокрутка программы
52	Плохое имя файла/плохой номер (Bad file name or number) Имя файла не соответствует требованиям DOS (например, не указан путь для файла не из текущего каталога)
53	Файл не найден (File not found) При попытке обращения к файлу неправильно указано его имя или путь к нему
54	Плохой режим файла (Bad file mod) Возникает, если файл создан в редакторе, не совместимым с редактором, используемым в настоящем случае
55	Файл уже открыт (File already open) Попытка повторного открытия файла или удаления открытого файла
56	Оператор FIELD активен (FIELD statement activ)
57	Ошибка в/вв устройства (Device I/O error) Ошибка устройства ввода/вывода, с которой не справляется DOS. Попробуйте посмотреть, все ли в порядке с аппаратной частью, т. е. с внешними устройствами компьютера
58	Файл уже существует (File already exists) Попытка сохранить файл под именем уже существующего на диске файла
59	Неверная длина записи (Bad record length)
61	Диск заполнен (Disk full) Диск, на который производится запись файла, не имеет достаточно места для этого. Надо освободить дисковое пространство, удалив что-нибудь менее важное
62	Ошибка: введен конец файла (Input past end of file) Возникает, когда для чтения из файла организуется цикл, число шагов которого больше числа содержащихся в файле записей. Чтобы избежать появления ошибки, используйте функцию `EOF` для досрочного выхода из цикла
63	Неверный номер записи (Bad record number)
64	Плохое имя файла (Bad file name) Имя файла не соответствует требованиям DOS
67	Слишком много файлов (Too many files)
68	Устройство недоступно (Device unavailable) В дисководе нет диска или он испорчен
69	Переполнение буфера коммуникации (Communication-buffer overflow) Попытка копирования в буфер слишком большого объема информации
70	Нет разрешения (Permission denied)
71	Ошибка формата диска (Disk not ready) Открыта защелка дисковода, в дисководе нет диска или он испорчен
72	Ошибка диска (Disk-media error) В дисководе нет диска или он испорчен

Таблица П2.1 *(окончание)*

Код	Сообщение об ошибке и возможная причина
73	Недоступная возможность (Advanced feature unavailable)
74	Переименование через диски (Rename across disks)
75	Ошибка доступа к пути/файлу (Path/File access error)
76	Путь не найден (Path not found) При попытке обращения к файлу неправильно указано его имя или путь к нему

ПРИЛОЖЕНИЕ 3

Примеры операторов

```
CLS
```

Оператор очистки экрана. Если все аргументы опущены, очищаются и графический, и текстовый экраны.

```
CLS 1
```

Очищает только графический экран, если он активен.

```
CLS 2
```

Очищает только текстовый экран, исключая нижнюю строку.

```
RANDOMIZE 32767 – TIMER
```

Оператор, запускающий генератор случайных чисел.

```
GOTO 20
```

Оператор безусловного перехода, здесь задает переход на строку 20.

```
END
```

Оператор-указатель конца программы, останавливает выполнение программы и закрывает все файлы.

```
SLEEP
```

Управляющий оператор, приостанавливающий выполнение программы до нажатия любой клавиши.

```
SLEEP 2
```

Управляющий оператор, приостанавливающий выполнение программы на 2 секунды.

```
REM Вся строка после оператора является комментарием
```

Оператор, указывающий на комментарий. Вместо ключевого слова `REM` можно использовать апостроф (`'`). При использовании `REM` в конце строки операторов его необходимо отделять двоеточием, а при записи с использованием апострофа двоеточие можно опускать.

```
PRINT 2*x ^ 2 + 1; (3 + 7) / (0.5 + 1.5); 3
```

Оператор при `x` = 2 вычислит и выведет на экран `9`, затем `5` и `3`.

```
PRINT "Значение x = "; 2; TAB(15); 3; SPC(5); 4
```

Оператор выведет на экран `Значение x = 2`, в 15-й позиции — `3` и, сделав 5 пробелов, выведет число `4`.

Примечание

1. Операторы в многооператорной строке разделяются двоеточием.
2. При наборе программы можно вместо слова PRINT ставить знак ?, который затем BASIC сам переделает в PRINT.
3. Если в программе впервые встречается какая-либо переменная, которой до этого не было присвоено никакого значения, то по умолчанию значение переменной будет равно 0.

```
READ x, y, z
```

Вводит числовые переменные x, y, z, для которых считывает из оператора DATA и присваивает числовые значения.

```
READ x, y$
```

Вводит числовую переменную x и строковую y$, для которых считывает из оператора DATA и присваивает x числовое значение, а y$ строковую константу.

```
DATA 1.2, -2.546, 3
```

Хранит числовые значения для последующего их считывания оператором READ.

```
DATA 7.2, "режим"
```

Хранит числовое значение и строковую константу для последующего их считывания оператором READ.

```
DATA 1.2, , 5, -4.8
```

Оператор с пустым элементом списка, присваивающим числовой переменной значение 0, а строковой — отсутствие значения.

```
RESTORE
```

Задает по умолчанию соответствующему оператору READ возврат на считывание с начала данных из первого оператора DATA.

```
RESTORE 50
```

Задает соответствующему оператору READ переход на считывание с начала данных из оператора DATA в строке 50 (50 — это метка).

Примечание

Переменные в операторе READ, как и данные в операторе DATA, разделяются запятыми.

Логические операторы BASIC приведены в *разд. 5.1* (см. табл. 5.2).

```
IF 0 < x THEN y = a * x + b ELSE y = c * x ^ 2
```

Линейная форма условного оператора. Удобна, если требуется реализовать два альтернативных условия. Если справедливо условие $0 < x$, то y определяется по формуле, стоящей после THEN, в противном случае — по формуле после ELSE.

```
IF 0 < x THEN y = a * x + b ELSE 50
```

Линейная форма условного оператора. Если справедливо условие $0 < x$, то вычисляется y после THEN, в противном случае выполняется переход на строку 50. После THEN или ELSE может стоять и многооператорная строка, например, `a = 2: b = 3: y = a * x + b: PRINT y`.

```
IF 0 < x THEN y = a * x + b
```

Неполная форма условного оператора. Если справедливо условие $0 < x$, то определяется y по формуле после THEN. В противном случае выполняется следующий оператор программы.

```
IF 0 < x AND x < 5 THEN y = a * x + b
```

Неполная форма условного оператора. Пример реализации сложного условия $0 < x < 5$ с помощью логического оператора AND.

```
IF 0 < x THEN IF x < 5 THEN y = a * x + b
```

Неполная форма условного оператора. Пример реализации сложного условия $0 < x < 5$ с помощью вложенного оператора IF.

```
IF x =< 0 THEN
y = a * x + b
ELSEIF 5 =< x THEN y = a * x
ELSEIF 0 < x AND x < 5 THEN y = c ^ x
ELSE PRINT "Такого варианта нет"
END IF
```

Блочная форма условного оператора. Блоки ELSE и ELSEIF могут быть опущены. Блоков ELSEIF может быть любое количество. Каждый из блоков может также содержать вложенные блочные структуры IF.

```
SELECT CASE n
CASE -1: PRINT "n может быть и отрицательным"
CASE 0: PRINT "n может быть числовым или строковым"
CASE 1, 2, 3: PRINT "Несколько одинаковых вариантов"
CASE 4 TO 8: PRINT "Использование ключевого слова TO"
CASE IS > 8: PRINT "Использование операций отношения"
CASE ELSE: PRINT " CASE ELSE может быть опущено"
END SELECT
```

Управляющий оператор, выполняющий один или несколько блоков оператора в зависимости от значения n (числового или строкового).

```
DIM m1(10), m2(1 TO 10, 5)
```

Оператор объявляет одномерный (m1) и двумерный (m2) массивы.

```
DIM m(10, 5)
DIM m(0 TO 10, 5)
DIM m(10, 0 TO 5)
DIM m(0 TO 10, 0 TO 5)
```

Операторы объявляют двумерный массив. При OPTION BASE 0 операторы эквивалентны. Ключевое слово TO позволяет указать нижнюю и верхнюю границы массива.

```
REDIM m(5, 3)
REDIM m(0 TO 5, 3)
REDIM m(5, 0 TO 3)
REDIM m(0 TO 5, 0 TO 3)
```

Оператор объявления массивов, изменяет объем памяти, выделенной динамическим массивам.

```
OPTION BASE n
```

Оператор объявления нижней границы массивов. Параметр n = 0 или 1, по умолчанию равен 0. Не обязателен, может использоваться только раз в модуле для описания всех массивов.

```
ERASE m
```

Оператор управления памятью, обнуляет элементы статического массива m или уничтожает динамический массив m.

```
FOR i = 2 TO 100 STEP 2
S = S + a( i ): IF S > 100 THEN EXIT FOR
NEXT i
```

Оператор цикла, реализующий суммирование элементов массива `a`, при сумме больше 100 предусмотрен досрочный выход из цикла.

```
FOR i = 1 TO 1000: NEXT i
```

Пустой цикл, реализующий временную задержку.

```
WHILE S <= 100
i = i +1: S = S + a( i )
WEND
```

Оператор цикла, реализующий суммирование элементов массива `a`. Цикл выполняется, пока указанное условие истинно (в данном случае при $S \le 100$).

Примечание

1. Возможные границы массивов находятся в приделах от –32 768 до 32 767.
2. Допускается вкладывать как циклы `FOR...NEXT`, так и `WHILE...WEND`.
3. Обычно при использовании циклов `FOR...NEXT` первому счетчику цикла дается имя `i`, вложенному в него — `j`, затем `k`, `l` и далее по алфавиту.

```
DO WHILE i > 10000
i = i +1
LOOP
```

Управляющий оператор цикла, повторяющий блок операторов, пока условие ложно, с проверкой условия в начале.

```
DO UNTIL INKEY$ <> ""
IF i > 10000 THEN EXIT DO ELSE i = i +1
LOOP
```

Управляющий оператор цикла, повторяющий блок операторов, пока условие истинно, с проверкой условия в начале. Производит увеличение значения счетчика `i` до тех пор, пока не будет нажата любая клавиша. Предусмотрен досрочный выход из цикла при значении счетчика `i` > 10 000.

```
DO
IF i > 10000 THEN EXIT DO ELSE i = i +1
LOOP WHILE INKEY$ = ""
```

Управляющий оператор цикла, повторяющий блок операторов, пока условие ложно, с проверкой условия в конце. Производит увеличение значения счетчика `i` до тех пор, пока не нажата любая клавиша. Предусмотрен досрочный выход из цикла при значении счетчика `i` > 10 000.

```
DO
i = i +1
LOOP UNTIL i < 10000
```

Управляющий оператор цикла, повторяющий блок операторов, пока условие истинно, с проверкой условия в конце.

```
PRINT "Нажмите клавишу ESC": DO: LOOP WHILE INKEY$ <> CHR$(27)
```

Пустой цикл, осуществляющий задержку до нажатия клавиши <Esc>.

`SWAP x( i ), x( j )`

Оператор, взаимно меняющий значения двух переменных.

`INKEY$`

Функция, читающая символы с клавиатуры. Если символ не был считан, то возвращается нулевая строка. Не дублирует ввод с клавиатуры выводом на экран (без "эха").

`CHR$(27)`

Символьная функция, возвращающая строку из одного символа, ASCII-код которого является аргументом, где код — число от 0 до 255. В данном случае аргумент — код клавиши <Esc>.

`GOSUB 300`

Управляющий оператор вызова подпрограммы на строке 300.

`RETURN 10`

Управляющий оператор выхода из подпрограммы, возвращающий на строку 10. Если метки нет, то управление возвращается на следующую за `GOSUB` строку.

`OPEN "file2.dat" FOR OUTPUT AS #2`

Оператор, открывающий по 2-му каналу файл file2.dat для создания. Если файл file2.dat уже был создан, то он уничтожается и создается заново.

`OPEN "file2.dat" FOR APPEND AS #1`

Оператор, открывающий по 1-му каналу файл file2.dat для добавления. Если файл находится не в текущем каталоге, с которым работает BASIC, то в названии файла необходимо указывать полный путь к нему, например, `"A:\ZZ\file.dat"` для файла, находящегося на диске A: (дискете) в каталоге ZZ.

`OPEN "file2.dat" FOR INPUT AS #3`

Оператор, открывающий по 3-му каналу файл file2.dat для считывания.

`WRITE #2, x10$ (1, j); x10$ (2, j)`

В открытый по 2-му каналу файл оператор записывает в одну строку из массива `x10$` два символьных элемента с разделительными знаками.

`PRINT #1, x10$ (1, j); x10$ (2, j)`

В открытый по 1-му каналу файл оператор записывает в одну строку из массива `x10$` два символьных элемента без разделительных знаков.

`INPUT #3, x11$ (i, j)`

Оператор, читающий данные из открытого по 3-му каналу файла и присваивающий их значения переменной `x11$(i,j)`.

`CLOSE #1`

Оператор, закрывающий открытый по 1-му каналу файл.

`CLOSE`

Если все аргументы опущены, то закрываются все файлы и устройства.

`EOF(n)`

Функция файлового ввода/вывода, определяющая условие конца файла, где `n` — номер открытого файла. Используется, например, для досрочного выхода из цикла, организованного для считывания данных из файла, когда данные файла исчерпаны.

`NAME " file1.dat" AS "file2.dat"`

Оператор переименовывает файл file1.dat в file2.dat.

```
KILL " file2.dat "
```

Оператор стирает указанный файл file2.dat с диска. Соответствует команде DOS `del`.

```
LINE (10, 15)-(200, 250), 1
```

Оператор рисует цветом 1 (синий) отрезок прямой линии из точки с координатами `x` = 10, `y` = 15 до точки с координатами `x` = 200, `y` = 250.

```
LINE (120, 180)-(290, 180), 1
```

Оператор рисует горизонтальную линию от `x` = 120 до `x` = 290 при `y` = 180.

```
LINE (10, 15)-(200, 270), 2, B
```

Оператор рисует цветом 2 (зеленый) прямоугольник со сторонами, параллельными экрану, диагональю которого является отрезок прямой из точки с координатами `x` = 10, `y` = 15 до точки с координатами `x` = 200, `y` = 270.

```
x1 = 10: y1 = 15: x2 = 200: y2 = 270
LINE (x1, y1)-(x2, y2), 2, BF
```

Оператор рисует прямоугольник, как в предыдущем случае, заливая его тем же цветом 2.

```
LINE -(40, 40), 1
```

Оператор рисует линию от последней точки, заданной предыдущим оператором программы (например, `x` = 10, `y` = 10), до точки `x` = 40, `y` = 40.

```
LINE -STEP(40, 40), 1
```

Оператор рисует линию от последней точки, заданной предыдущим оператором программы (например, `x` = 10, `y` = 10), до точки `x` = 50, `y` = 50.

```
LINE (20,20)-STEP(40, 40), 1
```

Оператор рисует линию от точки `x` = 20, `y` = 20 до точки `x` = 60, `y` = 60.

```
LINE STEP(20,20)-STEP(40, 40), 1
```

Оператор рисует линию (например, последняя точка `x` = 20, `y` = 20) от точки x = 40, `y` = 40 до точки `x` = 80, `y` = 80.

```
LINE STEP(20,20)-(60, 60), 1
```

Оператор рисует линию (например, последняя точка `x` = 10, `y` = 10) от точки `x` = 30, `y` = 30 до точки `x` = 60, `y` = 60.

```
LINE (120, 180)-(290, 180), 1, ,&H2A0F
```

Оператор рисует линию, стиль которой задается шестнадцатеричным числом `2A0F`.

```
LINE (120, 180)-(290, 280), 1, B,&H6C5F
```

Оператор рисует прямоугольник, стиль линии которого задается шестнадцатеричным числом `6C5F`.

Примечание

1. Стиль не влияет на закрашенные прямоугольники.
2. Если в операторе `LINE` не указан номер цвета, то он рисует основным цветом, а если указан 0 — фоновым.
3. Координаты и цвет в операторе `LINE` могут задаваться числами (например, 120), переменными (`x1`), элементами массива (`y(i,j)`), выражением (например, `(i + 2) / 3`).
4. Операторы `PSET` и `PRESET` работают одинаково, но если цвет не указан, `PSET` использует основной цвет, а `PRESET` — фоновый.
5. Если координаты точки находятся вне экрана, то никаких действий не производится и сообщение об ошибке не выдается.

`CIRCLE (200, 100), 50, 1`

Графический оператор, рисующий окружность с центром `x` = 200, `y` = 100, радиусом `r` = 50, цветом 1 (синий).

`CIRCLE STEP(100, 100), 20, 2`

Графический оператор, рисующий окружность со смещением центра относительно текущей позиции курсора, т. е. (пусть последняя точка `x` = 100, `y` = 0) с центром `x` = 200, `y` = 100, радиусом `r` = 20, цветом 2 (зеленый).

`CIRCLE (200, 100), 40, 3, 1.57, 4.71`

Графический оператор, рисующий дугу окружности с центром `x` = 200, `y` = 100, радиусом `r` = 40, цветом 3 (голубой). Число 1.57 задает начало дуги, 4.71 — конец.

`CIRCLE (200, 100), 40, 4, -1.57, -4.71`

Графический оператор, рисующий дугу окружности с центром `x` = 200, `y` = 100, и радиусы от концов дуги, радиусом `r` = 40, цветом 4 (красный). Число –1.57 задает начало дуги и рисование радиуса, –4.71 — конец дуги и рисование радиуса.

`CIRCLE (200, 100), 40, 5, 1.57`

Графический оператор, рисующий дугу окружности с центром `x` = 200, `y` = 100, радиусом `r` = 40, цветом 5 (фиолетовый). 1.57 задает начало дуги, 6.28 — конец (поскольку параметр опущен).

`CIRCLE (100, 200), 30, 3, , 4.71`

Графический оператор, рисующий дугу окружности с центром `x` = 100, `y` = 200, радиусом `r` = 30, цветом 6 (коричневый). 0 задает начало дуги (т. к. параметр опущен), 4.71 — конец дуги.

`CIRCLE (200, 100), 50, 7, , , 0.3`

Графический оператор, рисующий эллипс с центром `x` = 200, `y` = 100, радиусом `r` = 50, цветом 7 (серый) и сжатием по оси `y` (т. к. параметр < 1).

`CIRCLE (200, 100), 50, 7, , , 3`

Графический оператор, рисующий эллипс с центром `x` = 200, `y` = 100, радиусом `r` = 50, цветом 7 (серый) и сжатием по оси `x` (т. к. параметр > 1).

`PAINT (200, 100), 9, 8`

Графический оператор, закрашивающий цветом 9 (светло-синий) замкнутый контур, ограниченный линией цвета 8 (темно-серый), начиная с точки с координатами `x` = 200, `y` = 100.

`PAINT STEP(200, 100), 2, 1`

Графический оператор, закрашивающий цветом 2 (зеленый) замкнутый контур, ограниченный линией цвета 1 (синий), начиная с точки (например, последняя точка `x` = 100, `y` = 100) с координатами `x` = 300, `y` = 200.

Примечание

1. Последняя точка после завершения `CIRCLE` — центр окружности.
2. Если в операторе `CIRCLE` не указан номер цвета, то он рисует основным цветом, а если указан 0 — фоновым.
3. Дуги эллипсов рисуются аналогично дугам окружности.
4. Координаты и параметры в операторе `LINE` могут задаваться числами, переменными (`x1`), элементами массива (`y(i,j)`), выражением (например, `(i + 2) / 3`).

`DRAW` —

графический оператор, интерпретирующий символьное выражение и рисующий графический объект. Примеры команд оператора `DRAW` приведены в *разд. 8.1* (см. табл. 8.4 и рис. 8.6).

```
PALETTE 1, 2
```

Оператор установки цвета, изменяющий в палитре цвет 1 (синий) на цвет 2 (зеленый). Оператор без параметров восстанавливает начальные цвета палитры.

```
PALETTE USING PAL%(2)
```

Оператор изменяет цвета палитры в соответствии со значениями массива `PAL%`, начиная со 2-го элемента массива. Можно с его помощью переустановить цвета всей палитры.

```
PUT (100, 200), bug1, PSET
```

Графический оператор, рисующий на экране изображение, взятое в массив `bug1` оператором `GET`, начиная с точки с координатами `x` = 100, `y` = 200 (левый верхний угол прямоугольника). Переводит данные точка за точкой на экран.

```
PUT STEP(100, 10), bug1, PRESET
```

Графический оператор, рисующий изображение из массива `bug1`, начиная с точки (например, последняя точка `x` = 10, `y` = 20) с координатами `x` = 110, `y` = 30, выдавая инверсное изображение ("негатив").

```
PUT (20, 10), bug2, AND
```

Графический оператор, рисующий изображение из массива `bug2`, начиная с точки `x` = 20, `y` = 10. Изображение накладывается на существующую картинку. Точки, имеющие одинаковый цвет, сохраняются, остальные меняют цвет.

```
PUT (20, 10), bug2, OR
```

Графический оператор, рисующий изображение из массива `bug2`, начиная с точки `x` = 20, `y` = 10. Изображение накладывается на существующую картинку. Новое изображение не стирает предыдущего.

```
PUT (20, 10), bug2, XOR
```

Графический оператор, рисующий изображение из массива `bug2`, начиная с точки `x` = 20, `y` = 10. Изображение накладывается на существующую картинку. Действует так, что точки на экране, имеющие тот же цвет, что и образ, инвертируются. При повторном наложении образа предыдущая картинка восстанавливается. `XOR` устанавливается по умолчанию.

```
POINT(0)
```

Оператор возвращает координату `x` последней (текущей) точки.

```
POINT(1)
```

Оператор возвращает координату `y` последней (текущей) точки.

```
POINT (10, 20)
```

Оператор возвращает цвет точки (число) с координатами `x` = 10, `y` = 20.

```
GET (10, 10)-(120, 110), bug1
```

Графический оператор, считывающий изображение с прямоугольного участка экрана, с координатами верхней левой точки `x` = 10, `y` = 10 и нижней правой — `x` = 120, `y` = 110 в массив `bug1`.

```
GET STEP(10, 10)-STEP(20, 10), bug1
```

Графический оператор, считывающий изображение с прямоугольного участка экрана (пусть последняя точка `x` = 20, `y` = 10) с координатами верхней левой точки `x` = 30, `y` = 20 и нижней правой — `x` = 50, `y` = 30 в массив `bug1`.

Примечание

Если в программе использовался оператор `PALETTE` с параметрами, необходимо перед `END` поставить просто `PALETTE`.

Литература

1. Алиев В. К. Язык Бейсик. — М.: СОЛОН-Р, 2000.
2. Бобровский С. Программирование на языке QBasic для школьников и студентов. — М.: ДЕСС КОМ, Инфорком-Пресс, 2000.
3. Зельднер Г. А. QuickBasic. — М.: ABF, 1994.
4. Колесов Н. В., Ганитулин А. Х. Программирование на алгоритмических языках для ПЭВМ. Учебное пособие. — СПб.: СПбТИС, 1997.
5. Мельникова О. И., Бонюшкина А. Ю. Начала программирования на языке QBasic. — М.: ЭКОМ, 2000.
6. Очков В. Ф., Рахаев М. А. Этюды на языках QBasic, QuickBasic, Basic Compiler. — М.: Финансы и статистика, 1995.
7. Сафронов И. К. Бейсик в задачах и примерах. 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2006.
8. Федоренко Ю. Алгоритмы и программы на QBasic. Учебный курс. — СПб.: Питер, 2002.
9. Пестриков В. М., Маслобоев А. Н. Turbo Pascal 7.0. Изучаем на примерах. 2-е изд., перераб. и доп. — СПб.: Наука и Техника, 2004.